НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.89

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Програмний метод створення веб-сторінки за допомогою нейронної мережі на базі вхідного зображення»

Виконав:
студент ІІ курсу, групи КП-81мп
Богуцький Дмитро Борисович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент
Олещенко Любов Михайлівна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри АУТС ФІОТ, к.т.н., доцент,
Полторак Вадим Петрович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення» («Програмне забезпечення комп'ютерних та інформаційно-пошукових систем»)

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____ 2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Богуцькому Дмитру Борисовичу

1. Тема дисертації «Програмний метод створення веб-сторінки за допомогою нейронної мережі на базі вхідного зображення», науковий керівник дисертації доцент кафедри ПЗКС, к.т.н., Олещенко Любов Михайлівна затверджені наказом по університету від «13» листопада 2019 р. № 3895-с

2. Термін подання студентом дисертації «14» грудня 2019 р.

3. Об'єкт дослідження: процес розроблення веб-сторінки з використанням алгоритмів машинного навчання для генерації програмного коду веб-сторінки на базі вхідного зображення.

4. Предмет дослідження: сучасні алгоритми створення нейронних мереж та програмні засоби для автоматизації створення зовнішнього інтерфейсу веб-додатку.

5. Перелік завдань, які потрібно розробити:
   - провести аналіз методів гернерації коду за допомогою нейронних мереж;
   - дослідити роботу існуючих методів та алгоритмів створення веб-сторінок на базі вхідного зображення;
   - запропонувати та обґрунтувати шляхи модифікації методів створення веб-сторінок за допомогою нейронних мереж;
   - розробити власний метод генерації коду за допомогою нейронної мережі на базі вхідного зображення;
   - виконати програмну реалізацію розробленого методу;
   - виконати порівняння запропонованого методу із існуючими аналогами та надати оцінку отриманих результатів.

6. Орієнтовний перелік публікацій:
   - XII наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2019);

7. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., к.т.н., доцент		

8. Дата видачі завдання «15» грудня 2018 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.11.2018	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	04.12.2018	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	16.04.2019	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	14.05.2019	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	17.09.2019	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації; підготовка матеріалів доповіді на конференції ПМК-2019.	02.10.2019	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу;	28.10.2019	
8.	Оформлення текстової і графічної частини магістерської дисертації	10.11.2019	

Студент Д.Б. Богуцький

Науковий керівник дисертації Л.М. Олещенко

РЕФЕРАТ

Актуальність теми. На сьогоднішній день спостерігається стрімке зростання веб-сайтів, веб-застосунків та веб-сторінок в мережі Інтернет. На січень 2019 року офіційно зареєстровано 1.9 мільярдів активних веб-сайтів, що на півмільярда більше, ніж у минулому 2018 році. На кінець 2019 року прогнозується перехід відмітки до 2.6 мільярдів активних веб-сайтів. Якщо рахувати користувачів, то зараз в світі зареєстровано 5.11 мільярдів унікальних користувачів, що на 100 мільйонів, або на 2% більше, ніж у минулому році. Враховуючи наведену статистику, можна зрозуміти розмір IT-індустрії в нашому світі. Кожного дня створюється мільйони сайтів у всьому світі. Задача веб-розробника полягає у зміні існуючої функціональності сайту або у створені веб-сайту з нуля. Коли розробник створює веб-сайти з нуля кожен день, це зводиться до рутинної та не цікавої роботи. Також не всі люди мають потрібні знання для створення веб-сайтів. Виходячи з цієї проблеми, слід проаналізувати існуючі методи автоматизації даного процесу. Найкращим вибором технологічного стеку є використання нейронних мереж різних типів. На сьогоднішній день нейронні мережі дуже часто використовуються у нашому житті. На базі нейронних мереж створюють чат-ботів, автопілоти для машин, програми, що аналізують документи, розпізнавання зображень та багато іншого. Але, нажаль, у сфері генерації програмного коду існує дуже мало рішень. Таким чином, створення графічного інтерфейсу за допомогою нейронної мережі на базі вхідного зображення є актуальним завданням.

Об’єктом дослідження в даній роботі є процес розроблення веб-сторінки з використанням алгоритмів машинного навчання для генерації програмного коду веб-сторінки на базі вхідного зображення.

Предметом дослідження є сучасні алгоритми створення нейронних мереж та програмні засоби для автоматизації створення зовнішнього інтерфейсу веб-додатку.

Мета роботи полягає у розробленні програмного методу відтворення веб-сторінки за допомогою комбінованого використання нейронних мереж

GAN та RNN на базі вхідного зображення-шаблону, який дозволить усунути недоліки існуючих програмних рішень.

Методи дослідження. В роботі використовуються методи глибинного машинного навчання.

Наукова новизна роботи полягає в наступному. Запропонований програмний метод створення веб-сторінки за допомогою нейронної мережі на базі вхідного зображення відрізняється від існуючих методів створенням унікального дата-сету, завдяки якому дозволяється одночасна робота GAN та RNN, що, в свою чергу, робить алгоритм у середньому на 10% швидшим від аналогів та дозволяє генерувати адаптивний, семантичний та крос-браузерний програмний код веб-сторінки.

Практична цінність отриманих в роботі результатів полягає в тому, що запропонований метод дає змогу виправити недоліки існуючих методів генерації програмного коду шляхом комбінованого використання GAN і RNN. Завдяки цьому вдалося прискорити алгоритм, зробити код адаптивним, крос-браузерним та зменшити втрату пакетів при навчанні NN.

На основі розробленого методу розроблено веб-додаток на ReactJS, що дозволяє користуватися створеним методом та налаштовувати його.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на XII науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг», ПМК-2019 та опубліковані у збірнику тез доповідей.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень.

У першому розділі описана задача створення веб-сторінки за допомогою нейронних мереж різних типів, розглянуті особливості роботи існуючих

програмних рішень, оглянуті існуючі методи створення веб-сторінок, проаналізовані їх переваги та недоліки.

У другому розділі розглянуто принцип роботи нейронних мереж Елмана, проаналізовані її переваги та недоліки. Розглянуто існуючі методи машинного навчання. Запропоновано програмний метод на основі нейронної мережі Елмана для генерації програмного коду веб-сторінки.

У третьому розділі сформовані основні вимоги до програмного продукту; обґрунтовано вибір засобів, що використовувались для розробки; описана розроблена система, що реалізує власний метод одночасної роботи нейронних мереж різних типів для генерації програмного коду вихідної веб-сторінки.

У четвертому розділі визначено критерії оцінки ефективності, які застосовуються до розробленого методу; наведена інформація про дані, що використовувались при аналізі ефективності власного методу відносно показників існуючих рішень.

У висновках проаналізовано отримані результати роботи.

Робота виконана на 82 аркушах, містить 2 додатки та посилання на список використаних літературних джерел з 39 найменувань. У роботі наведено 24 рисунки, 7 таблиць та 5 лістингів.

Ключові слова: нейронні мережі, розпізнавання зображень, машинне навчання, генерація програмного коду, Python, ReactJS.

ABSTRACT

Actuality. Today, there is a rapid growth of websites, web applications and web pages on the Internet. As of January 2019, 1.9 billion active websites have been officially registered, which is half a billion more than last year. By the end of 2019, it is projected to hit the mark of 2.6 billion active websites. If you count users, there are now 5.11 billion unique users worldwide, up 100 million, or 2 percent more than last year. Given these statistics, you can understand the size of the IT industry in our world. Every day, millions of websites are created around the world. The task of the web developer is to change the existing functionality of the site or to create a website of 0. When a developer creates websites from 0 every day, it can lead to routine and not interesting work. Also, not all people have the knowledge to build websites. This is a problem that needs solving. Based on this problem, it is necessary to analyze the existing methods of automation of this process. The best choice for the technology stack is to use neural networks of different types. To date, neural networks have occupied no small part in our lives. Based on neural networks, they create chatbots, autopilots for cars, programs that analyze documents, image recognition and more. But unfortunately, in the area of code generation, there are very few solutions that can be counted on one hand. Thus, it is obvious that creating a graphical interface using a neural network based on the input image is an urgent task.

Object of research this work is a process of developing a web page using machine learning algorithms to generate webpage code based on the input image.

Subjects of research are modern neural network’s algorithms and software to automate the creation of the web application's external interface.

Goal of the work is the development of a software method of reproducing a web page using a combined use of GAN and RNN based on an input image template, which will eliminate the shortcomings of existing software solutions.

Methods of research. The methods of deep machine learning are used in the work

Scientific novelty The proposed software method of creating a web page using a neural network based on the input image differs from the existing methods

by creating a unique date set, which allows the simultaneous operation of GAN and RNN, which, in turn, makes the algorithm on average 10% faster than its counterparts, and allows you to generate responsive, semantic, and cross-browser web code.

Practical the results obtained are that the proposed method allows to correct the lack of existing methods of code generation through the combined use of GAN and RNN. This made it possible to speed up the algorithm, make the code responsive, cross-browser, and reduce packet loss in NN training.

Based on the developed method, an application on ReactJS has been developed that allows to use the created method and customize it.

Approbation. The main provisions and results of work were reported and discussed at the XII Scientific Conference of Masters and Postgraduate Students "Applied Mathematics and Computer", PMK-2019.

Structure and content of the thesis. The master's thesis consists of an introduction, four sections, conclusions and appendices.

The introduction gives a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and objectives of the research.

The first section describes the task of creating a web page using neural networks of different types, describes the features of existing software solutions, examines existing methods of creating web pages, their advantages and disadvantages are analyzed.

The second section deals with the principle of operation of the Elman neural networks, its advantages and disadvantages are analyzed. Existing machine learning methods are considered. A proprietary Elman-based neural network method for generating web page code is proposed.

In the third section the basic requirements for the software product are formed; the choice of the means used in the development is justified; describes a developed system that implements its own method of simultaneous operation of neural networks of different types to generate the source code of the source web page.

The fourth section defines the performance evaluation criteria that apply to the method developed; provides information on data used in performance analysis; an analysis of the effectiveness of its own method against the indicators of existing solutions.

The results of the work are analyzed in the conclusions.

The work is made on 82 sheets, contains 2 appendices and links to the list of used literary sources of 39 titles. The paper presents 24 figures, 7 tables and 5 listings.

Keywords: neural networks, image recognition, machine learning, programming code generation, Python, ReactJS.

РЕФЕРАТ

Актуальность. Сегодня наблюдается быстрый рост веб-сайтов, веб-приложений и веб-страниц в Интернете. По состоянию на январь 2019 года было официально зарегистрировано 1,9 миллиарда активных веб-сайтов, что на полмиллиарда больше, чем в прошлом году. По прогнозам, к концу 2019 года он достигнет отметки в 2,6 миллиарда активных веб-сайтов. Если вы посчитаете пользователей, то сейчас в мире насчитывается 5,11 миллиарда уникальных пользователей, что на 100 миллионов или на 2 % больше, чем в прошлом году. Учитывая эти статистические данные, вы можете понять масштабы ИТ-индустрии в нашем мире. Каждый день миллионы веб-сайтов создаются по всему миру. Задача веб-разработчика состоит в том, чтобы изменить существующую функциональность сайта или создать веб-сайт с нуля. Когда разработчик создает веб-сайты с нуля каждый день, это может привести к рутинной и не интересной работе. Кроме того, не все люди имеют знания для создания сайтов. Исходя из этой проблемы, необходимо проанализировать существующие методы автоматизации этого процесса. Наилучшим выбором для технологии стека является использование нейронных сетей разных типов. На сегодняшний день нейронные сети занимают немалую часть в нашей жизни. На основе нейронных сетей они создают чат-ботов, автопилоты для автомобилей, программы для анализа документов, распознавания изображений и многое другое. Но, к сожалению, в области генерации кода очень мало решений. Таким образом, очевидно, что создание графического интерфейса с использованием нейронной сети на основе входного изображения является актуальной задачей.

Объектом исследования в данной работе является процесс разработки веб-страницы с использованием алгоритмов машинного обучения для генерации программного кода веб-страницы на базе входного изображения.

Предметом исследования являются современные алгоритмы создания нейронных сетей и программные средства для автоматизации создания внешнего интерфейса веб-приложения.

Целью исследования является разработка программного метода воспроизведения веб-страницы с помощью комбинированного использования GAN и RNN на базе входного изображения-шаблона, который позволит устранить недостатки существующих программных решений.

Методы исследования. В работе используются методы глубинного машинного обучения.

Научная новизна работы заключается в следующем.

Предложенный программный метод создания веб-страницы с помощью нейронной сети на базе входного изображения отличается от существующих методов созданием уникального дата-сета, благодаря которому разрешается одновременная работа GAN и RNN, что, в свою очередь, делает алгоритм в среднем на 10% быстрее от аналогов и позволяет генерировать адаптивный, семантический и кросс-браузерный программный код веб-страницы.

Практическая ценность полученных в работе результатов заключается в том, что предложенный метод позволяет исправить недостаток существующих методов генерации программного кода путем комбинированного использования GAN и RNN. Благодаря этому удалось присокриты алгоритм, сделать код адаптивным, кросс-браузерный и уменьшить потерю пакетов при обучении NN.

На основе разработанного метода разработаны приложение на ReactJS, что позволяет пользоваться созданным методом и настраивать его.

Апробация работы. Основные положения и результаты работы докладывались и обсуждались на XII научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2019.

Структура и объем работы. Магистерская диссертация состоит из введения, четырех глав, заключения и приложений.

Во введении дана общая характеристика работы, выполнена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследований.

В первой главе описана задача создания веб-страницы с помощью нейронных сетей различных типов, рассмотрены особенности работы существующих программных решений, осмотрены существующие методы создания веб-страниц, проанализированы их преимущества и недостатки.

Во втором разделе рассмотрены принцип работы нейронных сетей Элмана, проанализированы ее преимущества и недостатки. Рассмотрены существующие методы машинного обучения. Предложенный программный метод на основе нейронной сети Элмана для генерации программного кода веб-страницы.

В третьем разделе сформированы основные требования к программному продукту; обоснован выбор средств, которые использовались при разработке; описана разработанная система, реализующая собственный метод одновременной работы нейронных сетей различных типов для генерации кода исходной веб-страницы.

В четвертом разделе определены критерии оценки эффективности, которые применяются к разработанного метода; приведена информация о данных, которые использовались при анализе эффективности; проведен анализ эффективности собственного метода показателей существующих решений.

В выводах проанализированы полученные результаты работы.

Работа выполнена на 82 листах, содержит 2 приложения и ссылки на список использованных литературных источников из 39 наименований. В работе приведены 24 рисунка, 7 таблиц и 5 листингов.

Ключевые слова: нейронные сети, распознавание изображений, машинное обучение, генерация программного кода, Python, ReactJS.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

– інструмент розробки програмного забезпечення, котрий спрощує створення графічного інтерфейсу користувача.

– стандартна мова розмітки для створення веб-сторінок і веб-додатків.

– мова, розроблена компанією «ABBYY» для розмітки словникових статей відповідно до технології показу словників, прийнятої в системі електронних словників «Lingvo».

– колекція однотипних даних, що застосовується в задачах машинної обробки даних.

– засіб зручної взаємодії користувача з інформаційною системою.

– є класом алгоритмів штучного інтелекту, що використовуються в навчанні без учителя.

– це клас глибинних штучних нейронних мереж прямого поширення, який успішно застосовувався до аналізу візуальних зображень.

Фреймворк – це реальна або концептуальна структура, призначена слугувати опорою або керівництвом для побудови чогось, що розширює структуру на щось корисне.

API (Application Programming Interface) – це набір публічних методів та властивостей, які використовуються для взаємодії з іншими об’єктами у програмі.

ВСТУП

Задача створення алгоритму генерації програмного коду на базі вхідного зображення є важливою як у сфері глибинного машинного навчання, так і в сфері інформаційних технологій, особливо для розпізнавання зображень і зводиться не лише до звичайної інженерної задачі, що має на даний час дуже мало готових рішень, але й вирішує великий обсяг задач та має наукову новизну. В першу чергу, це стосується існуючих обмежень щодо методів навчання нейронних мереж та способів машинного навчання, при яких результат роботи однієї нейронної мережі використовується як база для роботи іншої.

Для створення алгоритму розпізнавання зображення та генерації програмного коду зазвичай використовують одну єдину нейронну мережу. Таким чином, для розпізнавання зображення, створення датасету, дерева обходу та генерації програмного коду використовують генеративно-змагальну нейронну мережу. Цей спосіб має ряд недоліків, головним з яких є неспроможність масштабуватися та прискорювати час обробки зображень.

Алгоритми генерації програмного коду такого типу мають віддавати всі процеси обробки інформації на віддалений сервер, таким чином, збільшуєтсья час роботи всього алгоритму та створюється потреба в постійному контролі процесу обробки, начання нейронної мережі та якості вихідного програмного коду. Однією з фундаментальних проблем розпізнавання зображень та генерації програмного коду є велике навантаження, та те, що віддалені сервери мають свої обмеження в потужності та обчислювальної можливості. Саме наведені вище обмеження стали ключовим фактором в стагнації існуючих алгоритмів та створенні нових. Відсутність користувацького інтерфейсу, постійні змінни методів розробки сучасних веб-сторінок не дають можливості розробникам поліпшувати існуючі алгоритму генерації програмного коду та створювати нові рішення та методи.

Немає однієї системи або інноваційного рішення, що виправить дану ситуацію. Робота існуючих алгоритмів залежить від десятків факторів, через які розробникам потрібно постійно вдосконалювати процеси обробки дата-сетів, способи генерації програмного коду та багато іншого.

Безпосередньо система, як було зазначено вище, не має навіть мінімального користувацького інтерфесу, що не дає змогу користувачам користуватися алгоритмом, або хоча б побачити, як він працює. Це створює нові наукові проблеми, такі, як розроблення нових методів генерації програмного коду, створення інтерфейсу, зменешення навантаження на віддалений сервер, простота у використанні та швидка адаптація до нових методів генерації веб-сторінок.

Враховуючи все вище зазначене, очевидно, що дана проблематика є доволі перспективною в рамках методів генерації програмного коду, що на фоні нових методів машинного навчання може дати гарні результати в імплементуванні розроблених рішень у цій сфері.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз існуючих рішень розпізнавання текстів

Існує кілька основних методів розпізнавання тексту. Всі вони є комерційними продуктами, і багато внутрішніх алгоритмів їх роботи приховані від загального доступу. Принцип роботи подібних систем базується на кількох стратегіях, але найчастіше, алгоритм розпізнавання у загальнішому вигляді полягає в послідовному висуванні і перевірці гіпотез [1]. При цьому порядок їх висунення регулюється закладеними в програму знаннями про досліджуваний предмет і результатами перевірки попередніх гіпотез.

1.1.1. *Розпізнання за допомогою метрик*

Метрика – деяке умовне значення функції, що визначає положення об'єкта в просторі. Таким чином, якщо два об'єкти розташовані близько один від одного, тобто схожі (наприклад, дві літери А написані різним шрифтом), то метрики для таких об'єктів будуть збігатися або бути гранично схожими. Для розпізнавання в цьому режимі була обрана метрика Хеммінга.

Метрика Хеммінга – метрика яка показує, як сильно об'єкти не схожі між собою.

Дану метрику часто використовують при кодуванні інформації та передачі даних. Наприклад, після сеансу передачі на виході є наступна послідовність біт (1001001), також нам відомо, що повинна прийти інша послідовність біт (1000101) [2]. Ми обчислюємо метрику шляхом порівняння частин послідовності з відповідними місцями з іншої послідовності. Таким чином, метрика Хеммінга в нашому випадку дорівнює 2, так як об'єкти відрізняються в двох позиціях (2 – це ступінь відмінності, чим більше, тим гірше в нашому випадку).

Отже, щоб визначити, яка буква зображена, потрібно знайти її метрику з усіма готовими шаблонами. І той шаблон, чия метрика виявиться найбільш близькою до 0, буде відповіддю.

Як показала практика, підрахунок однієї лише метрики не дає позитивного результату, так як багато літер схожі між собою, наприклад «j», «i», що призводить до помилкового розпізнавання.

Тоді було прийнято рішення придумати нові метрики, що дозволяють розмежувати деяку множину літер в окремий клас. Зокрема, були реалізовані метрики відображення горизонтального і вертикального, переважання ваги горизонтального і вертикального.

Експериментально було з'ясовано, що такі літери як «H», «I», «i», «O», «o», «X», «x», «l» мають суперсиметрією (повністю збігаються зі своїми відображеннями і значущі пікселі розподілені рівномірно по всьому зображенню), тому вони були винесені в окремий клас, що скорочує перебір всіх метрик приблизно в 6 разів [3]. Аналогічні дії були проведені відносно інших букв. В середньому зменшення перебору досягає приблизно 3 рази.

Також є унікальна буква така як «J», яка знаходиться в своєму класі одна, і значить, ідентифікується однозначно. Далі, для кожного класу вираховується метрика Хеммінга, яка на даному етапі дає кращі показники, ніж при прямому застосуванні.

При створенні шаблонів використовувався шрифт «consolas», тому, якщо розпізнається текст, написаний цим шрифтом, розпізнавання має точність близько 99 відсотків. При зміні шрифту, точність падає до 70 відсотків.

1.1.2. *Розпізнавання за шаблоном*

Програмне забезпечення OCR (Optical Character Recognition – оптичне розпізнавання символів), як правило, з великим растровим зображенням сторінки з сканера. При цьому більшість систем має шаблони, створені для різних накреслень. Після кількох розпізнаних слів, програмне забезпечення

визначає використовуваний шрифт і шукає відповідні пари тільки для цього шрифту. В деяких випадках програмне забезпечення використовує чисельні значення елементів символу (пропорцій), щоб визначити новий шрифт. Це може поліпшити ефективність розпізнавання. Програма розпізнавання TypeReader використовує машинно-залежні алгоритми на основі шаблонного підходу. Даний підхід вимагає створення шаблону для кожного шрифту. Наприклад, програма TypeReader використовує 2100 різних варіантів накреслень символів.

1.1.3. *Розпізнавання за допомогою саморозвитку*

В математичному сенсі нейронна мережа – це лише модель біологічного визначення.

Існують також багато різновидів цих моделей. У своїй роботі я використовував одношарову мережу Кохонена.

Принцип роботи нейронної мережі такий, що, навчивши на вхідному шарі нейрони, нове зображення мережі реагує імпульсом того чи іншого нейрона. Всі нейрони приймають значення букв. Кожен нейрон крім виходу має також безліч входів. Дані входи описують значення пікселя зображення [4]. Тобто, якщо є зображення 16x16, входів у мережі має бути 256.

Кожному входу присвоюється певний коефіцієнт, по закінченню розпізнавання на кожному нейроні накопичується певний заряд. У якого нейрона заряд буде більше, той нейрон і випустить імпульс.

Для того, щоб коефіцієнти входів були правильно налаштовані, необхідно спочатку навчити мережу. Цим займається окремий модуль навчання. Даний модуль бере чергове зображення з навчальної вибірки і віддає його мережі. Мережа аналізує всі позиції чорних пікселів і вирівнює коефіцієнти, мінімізуючи помилку збігу методом градієнта, після чого певні нейрони зіставляють дане зображення.

Приклад навчання:

```
public void Teach(Bitmap img, Neuron correctNeuron)
            {
                var vector = GetVector(img);
                for (int i = 0; i < vector.Length; i++)
                {
                    vector[i] *= 10;
                    correctNeuron.Weigths[i] = correctNeuron.Weigths[i] + 0.5 * (vect
or[i] - correctNeuron.Weigths[i]);
                }
             }
```

Рис. 1. Приклад навчання нейронної мережі

По закінченню навчання кожен нейрон схожий на полотно художника, де на місцях, в яких найчастіше зустрічалися чорні пікселі – найбільш темна фарба (значення заряду більше), а там, де рідше – зовсім світлий тон.

Рис. 2. Відхилення нейронів при розпізнавання літери

Всі коефіцієнти вирівняні і готові сприймати зображення. Точність розпізнавання при цьому методі досягає 80 відсотків. Слід зауважити, що точність розпізнавання залежить від навчальної вибірки, як від кількості, так і від якості.

1.1.4. *Структурний підхід*

Набільш відома світі система OCR – Caere OmniPage Professional використовує алгоритм, заснований на знаходженні загальних специфічних особливостей символів [5]. Ця система містить 100 різних алгоритмів для ідентифікації 100 різних символів: верхнього і нижнього регістра від «A» до «Z», записи чисел і символів пунктуації. Кожен з цих алгоритмів шукає «Особливості» накреслень типу «островів», «півостровів», точок, прямих відбитків і дуг. Експертні системи також розглядають горизонтальні і вертикальні проекції відбитків букви і звертають увагу на основні особливості в створених кривих, підсумовуючи в них число темних пікселів. На жаль, нечіткий текст може стати специфічною проблемою для цих структурних алгоритмів, так як відсутній піксель може розбивати довгий штрих чи криву, а додаткова пляма бруду може закривати петлю.

1.1.5. *Контекстне розпізнавання*

У програмне забезпечення системи OCR часто включається словники для допомоги алгоритмам розпізнавання. Словники надають довідки у багатьох випадках, але швидко відмовляють, коли, наприклад, мають справу з іменами власними, які не перебувають у словнику [6]. Цей ефект особливо помітний в російській програмі FineReader, який частіше, ніж в середньому за всіма символами, помиляється в словах, які відсутні в його словнику.

1.2. Аналіз існуючих комерційних програмних продуктів

На даний момент велика кількість комерційних мережевих продуктів використовують в своїх рішеннях алгоритми машиного навчання.

Ці та багато інших видів послуг об’єднує необхідність постачати клієнтові постійне та якісне з’єднання до мережі, необхідність синхронізувати дані та зберігати стани не тільки в одному місці, але й в багатьох місцях одночасно.

Програмні продукти так чи інакше завжди використовують певну програмну реалізацію мережевого балансування та маршрутизацію, адже це

є основою роботи будь-якої мережі [7]. Далі неведено декілька прикладів програм.

1.2.1. *Pix2Code*

Датський стартап Uizard Technologies представив нейромережу pix2code, яка генерує готовий код на базі зображення користувацького інтерфейсу. Точність розпізнавання вхідного шаблону становить приблизно 77%.

Рис 3. Робота нейронної мережі pix2code

Функціональне ядро нейронної мережі використовує унікальні алгоритми, розроблені командою інженерів, що забезпечує:

- Тренування нейронної мережі.
- Форматування файлу.
- Створення файлів-ключів на певний файл.
- Створення дата-сету.
- Відтворення макету.
- Виконання великої кількості ітерацій.
- Можливість функціонування мережі без основного сервера.

Training data represented in words

Input sentence	nothing, nothing, nothing, nothing, <start token>	**Input images**	screenshot.jpg
	nothing, nothing, nothing, <start token>, <HTML>		screenshot.jpg
	nothing, nothing, <start token>, <HTML>, Hello World!		screenshot.jpg
	nothing, <start token>, <HTML>, Hello World!, </HTML>		screenshot.jpg
Output	<HTML>		
	Hello World!		
	</HTML>		
	<end token>		

Training data represented in digits

Input sentence	[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [1, 0, 0, 0, 0]]	**Input images**	[pixel values]
	[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [1, 0, 0, 0, 0], [0, 1, 0, 0, 0]]		[pixel values]
	[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 1, 0, 0]]		[pixel values]
	[[0, 0, 0, 0, 0], [1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 1, 0, 0], [0, 0, 0, 1, 0]]		[pixel values]
Output	[0, 1, 0, 0, 0]		
	[0, 0, 1, 0, 0]		
	[0, 0, 0, 1, 0]		
	[0, 0, 0, 0, 1]		

Рис. 4. Тренування нейронної мережі pix2code

Стратегічним завданням є тренування алгоритму передбачувати відповідні теги, послідовно переглядаючи зображення на вході. Коли алгоритм передбачає наступний тег, він отримує зображення та всі правильні теги для цього елементу.

1.2.2. *Sketching Interfaces*

Scetching interfaces – це стартап команди Airbnb. Час, необхідний для тестування ідеї, повинен бути нульовим [8]. Команда вірить, що впродовж найближчих років нові технології дозволять командам розробляти нові продукти виразно та інтуїтивно, одночасно усуваючи перешкоди від процесу розроблення продукту.

На сьогоднішній день кожен крок у процесі проектування і кожен вироблений артефакт може не мати розвитку. Робота припиняється, коли одна дисципліна закінчує частину проекту і передає відповідальність на іншу дисципліну. Проекти прогресу від засідань зацікавлених сторін до проектування до інженерії; вимоги стають дослідженнями, дослідження

стають макетами і прототипами, і вони передаються розробникам, щоб стати кінцевими продуктами. Але кожна з цих громіздких кроків, по суті, є перекладом загального значення на інший носій у прогресії до спільної мети, з кваліфікованими фахівцями в кожній області, які виконують функції перекладачів.

Для спрощення цього процесу команда AirBnb почала вивчати методи, щоб привести час тестування до нуля.

Ескіз здавався природним місцем для початку. Як дизайнери інтерфейсу, ескіз – це інтуїтивний метод вираження концепції. Команда хотіла побачити, як це може виглядати, щоб пропустити кілька кроків у життєвому циклі розроблення продукту і миттєво перевести ескізи в готовий продукт.

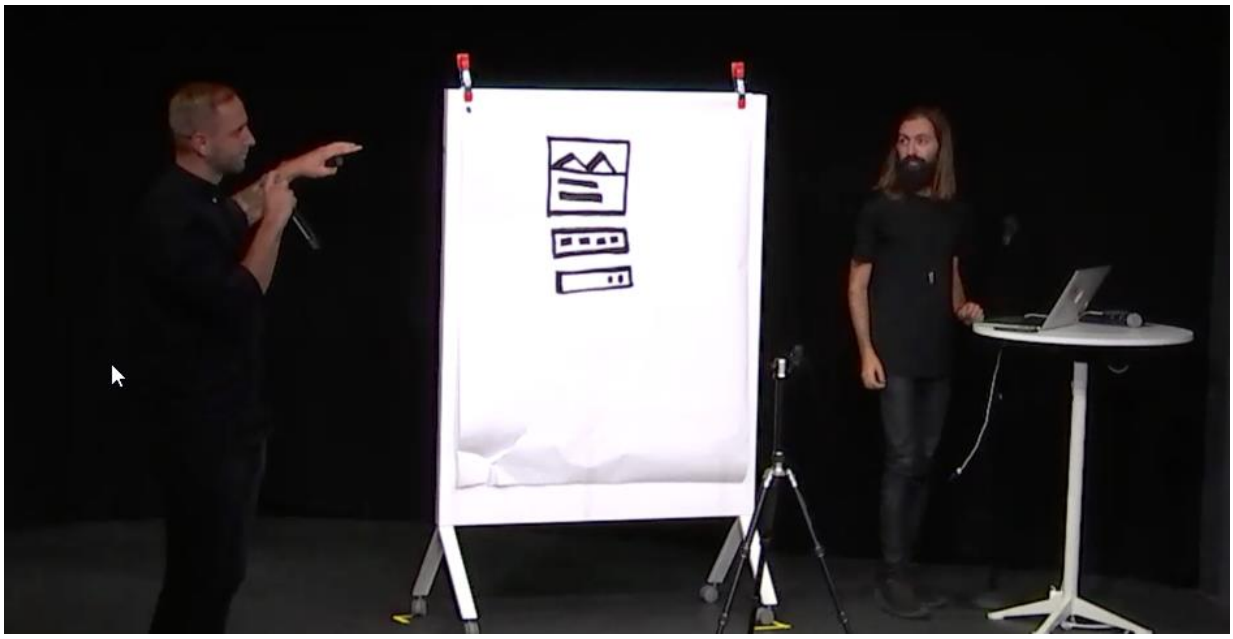


Рис. 5. Демонстрація команди Scetching interfaces

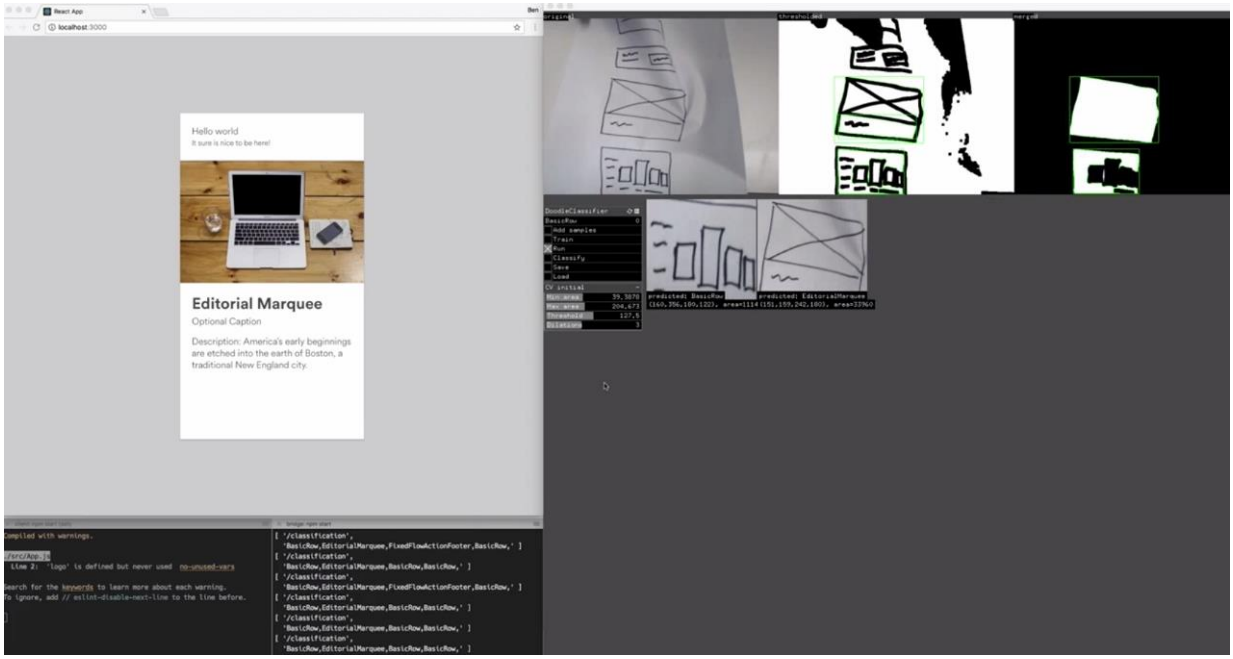


Рис. 6. Демонстрація роботи алгоритму Scetching interface

Ця система вже продемонструвала великий потенціал. Команда експериментувала з використанням технологій для прототипів живого коду з малюнків на дошці, щоб перекласти зображення з високою точністю у специфікації компонентів для інженерів, а також перевести виробничий код у файли дизайну для ітерації дизайнерами.

1.2.3. *Deep Systems*

Команда Deep Systems працює над розробкою великої кількості рішень на тему комп'ютерного зору, наприклад, безпілотні автомобілі, систему розпізнавання квитанцій, виявлення дефектів на дорогах і т.д. Фахівці витрачають багато часу на роботу з даними для навчання: створення анотацій зображень, об'єднання даних з загальнодоступними наборами даних, доповнення даних [9]. Supervisely спрощує роботу з навчальними даними і автоматизує багато рутинних завдань. Розуміння архітектури нейронної мережі є ключовим.

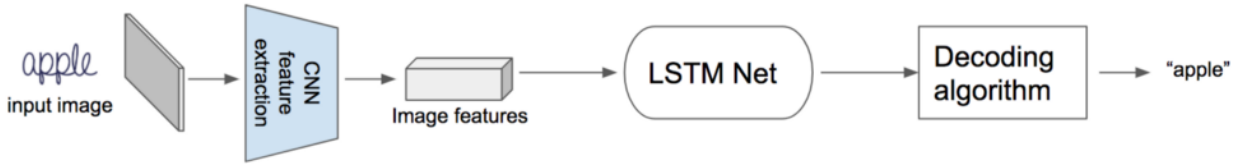


Рис. 7. Архітектура deep systems

По-перше, зображення завантажується в згорткову нейронну мережу для вилучення функцій зображення. Наступним кроком буде застосування рекурентної нейронної мережі для цих функцій, за якою слідує спеціальний алгоритм декодування [10]. Цей алгоритм декодування приймає lstm виходи з кожного часового кроку і виробляє остаточне маркування.

Далі показана докладна архітектура. FC – повністю підключений шар, SM – шар softmax.

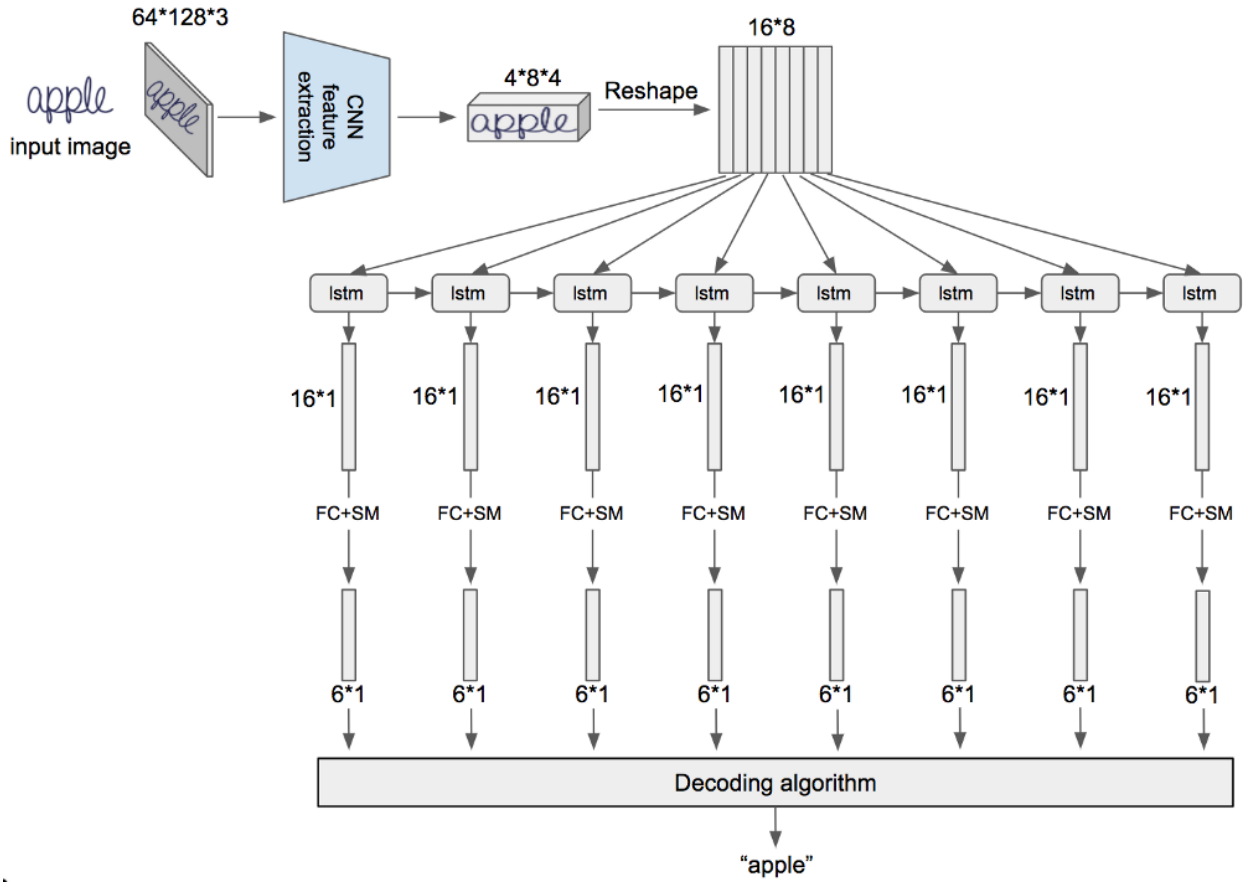


Рис. 8. Архітектура deep systems

Зображення має наступні розміри: висота дорівнює 64, ширина дорівнює 128 і кількість каналів дорівнює трьом.

Ми подаємо тензор зображення на екстрактор ознак згорткової нейронної мережі, і він створює тензор з розмірами 4 * 8 * 4. Ми поміщаємо зображення apple в тензор об'єктів, щоб зрозуміти, як його інтерпретувати. Висота дорівнює 4, ширина дорівнює 8 (це просторові розміри), а число каналів дорівнює 4. Таким чином, ми перетворимо вхідне зображення з 3-ма каналами в 4-канальний тензор [11]. На практиці кількість каналів має бути набагато більше.

Потім ми повторюємо операцію. Після цього ми отримуємо послідовність з 8 векторів з 16 елементів. Після цього ми подаємо ці 8 векторів в мережу LSTM і отримаємо її вихід – також вектори з 16 елементів. Потім ми застосовуємо повністю пов'язаний шар, за яким слідує шар softmax, і отримуємо вектор з 6 елементів [12]. Цей вектор містить розподіл ймовірностей розпізнавання символів алфавіту на кожному кроці LSTM.

На практиці число вихідних векторів нейронної мережі може досягати 32, 64 або більше. Вибір буде залежати від конкретного завдання. Також у виробництві краще використовувати багатошарову двосторонню LSTM. Але цей простий приклад пояснює тільки найважливіші поняття.

Але як працює алгоритм декодування? На наведеній вище діаграмі ми маємо вісім векторів ймовірностей на кожному кроці часу LSTM. Візьмемо найбільш ймовірний символ на кожному часовому кроці. В результаті ми отримуємо рядок з восьми символів – одну найбільш ймовірну букву на кожному часовому кроці. Потім ми повинні склеїти усі послідовні повторювані символи в один. У нашому прикладі дві букви «е» приклеєні до однієї. Спеціальний порожній символ дозволяє розділити знаки, які повторюються в оригінальному маркуванні. Ми додали порожній символ в алфавіт, щоб навчити нашу нейронну мережу передбачити пробіл

між такими символами. Потім ми видалимо всі порожні символи. Подивимося на ілюстрацію нижче.

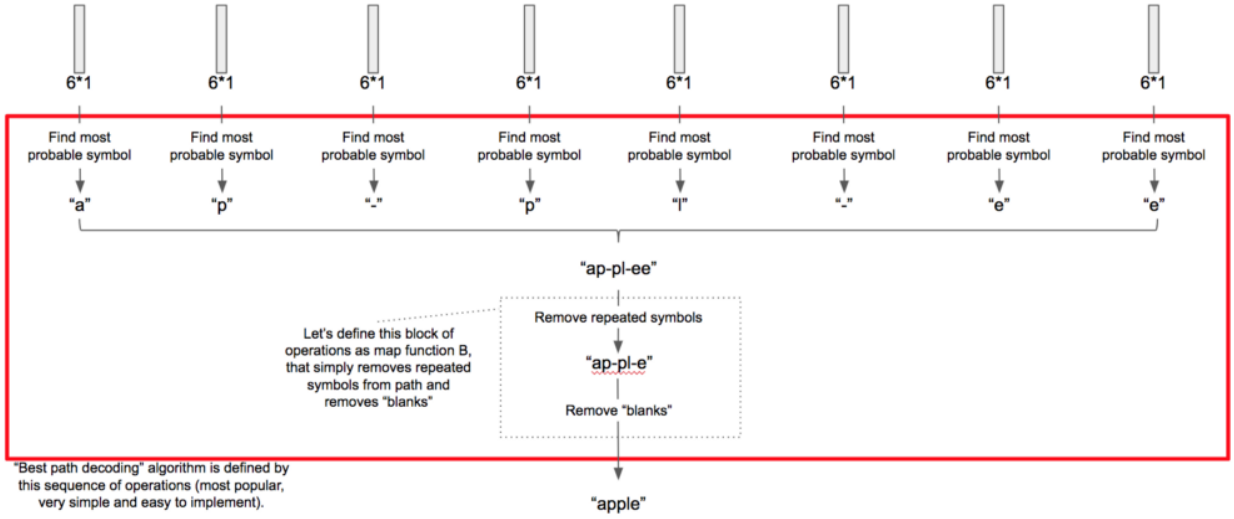


Рис. 9. Розпізнавання слів

Коли ми тренуємо мережу, ми замінюємо алгоритм декодування шаром CTC-Loss [13].

В даній реалізації використовується складніша архітектура нейронної мережі, але основні принципи тут все ті ж.

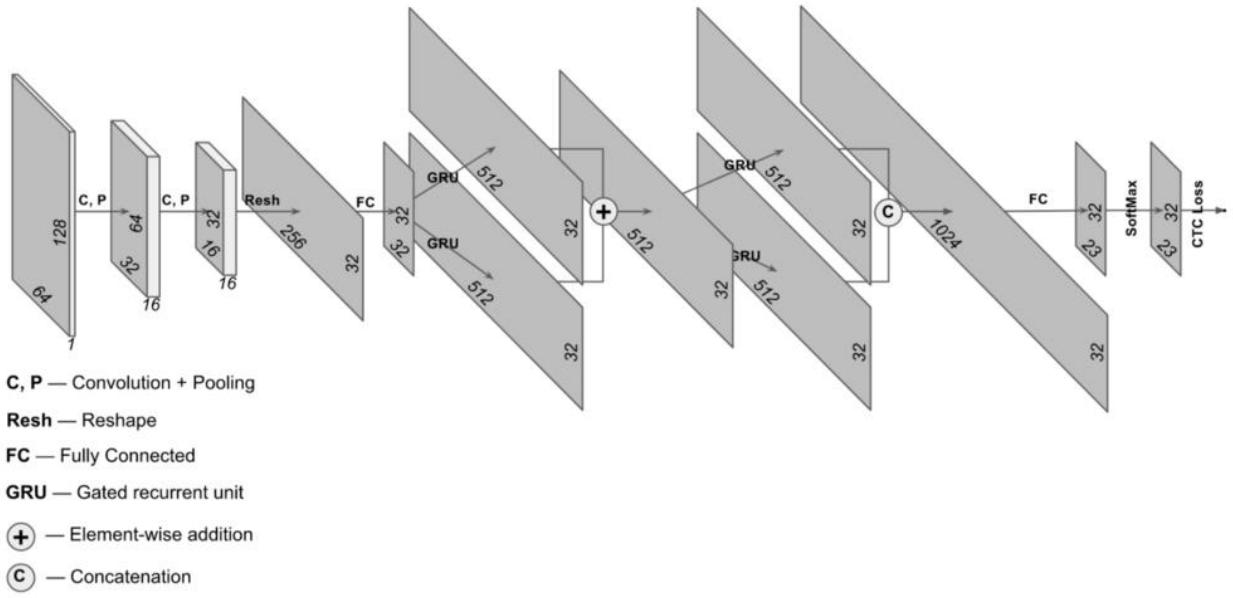


Рис. 10. Шари даних deep systems

Після навчання моделі ми застосовуємо її на зображеннях з тестового набору і отримуємо дуже високу точність. Ми також візуалізуємо розподіл ймовірностей з кожного кроку рекурентної нейронної мережі у вигляді матриці (рис. 11).

Рис. 11. Розпізнавання тексту на автомобільному номері

Рядки цієї матриці відповідають всім символам алфавіту і пробілу. Стовпці відповідають крокам нейронної мережі.

1.3. Висновки до розділу 1

Задача побудови нейронної мережі для розпізнавання зображення та генерації готового коду є надзвичайно важливою задачею як у сфері машинного навчання, так і в загальній інформаційно-обмінній сфері і зводиться не лише до тривіальної інженерної задачі, що має в собі вже на даний час багато рішень, а зводиться до побудови нейронної мережі для роботи над недоліками існуючих алгоритмів, покращенню точності вимірювань та прискорення швидкодії. Основними недоліками існуючих алгоритмів є необхідність у технічних навичках для користування алгоритмом, відсутність користувацького інтерфейсу, відсутність

налаштувань та необхідність в дотримуванні спеціальних вимог для створення вхідного зображення-шаблону.

На основі наведених недоліків існуючих рішень, виділено наступні функіональні можливості, які треба реалізувати у власному програмному методі:

- Користувацький інтерфейс.
- Гнучке налаштування роботи методу.
- Розпізнавання зображення, що зроблене у будь-якому редакторі.
- Збільшити загальну швидкодію роботи програмного забезпечення.

2. ФОРМУЛЮВАННЯ ПРОГРАМНОГО МЕТОДУ ВІДТВОРЕННЯ ВЕБ-СТОРІНКИ З ВХІДНОГО ЗОБРАЖЕННЯ

2.1. Аргументація вибору штучної рекурентної нейронної мережі

В результаті розгляду різноманітних видів і типів штучних нейронних мереж виявлено, що їх можна поділити на 3 головні категорії:

- Згорткові.
- Рекурентні.
- Нейронна мережа Елмана.

Основними елементами структури нейронної мережі є:

- Штучні нейрони.
- Синапс.
- Сигнал.

Завдяки цим характеристикам можна вирішити, який саме вид та тип нейронної мережі потрібен для створення програмного методу.

В ході розроблення та в рамках підтримки програмного методу треба базуватися на швидкодії нейронної мережі та на зручності взаємодії користувача з інтерфейсом програмного продукту. Рекурентні нейронні мережі, в свою чергу, утворюють орієнтовний цикл між з’єднанням нейронів. Також інформація у алгоритмі повинна передаватись і між самими нейронами [14].

Для того, щоб розробити програмний метод, що використовує одночасну роботу двох нейронних мереж, слід проаналізувати, які нейронні мережі можуть доцільно працювати з іншими.

Під даний опис підпадають декілька нейронних мереж, а саме згорткова нейронна мережа та рекурентна згорткова мережа. Для реалізації методу відтворення веб-сторінки з вхідного зображення мною були обрані рекурентні нейронні мережі Елмана і Джордана. Вони використовують

тришарову мережу з додованням набору “контекстних вузлів”. Архітектура мережі наведена на рис. 12.

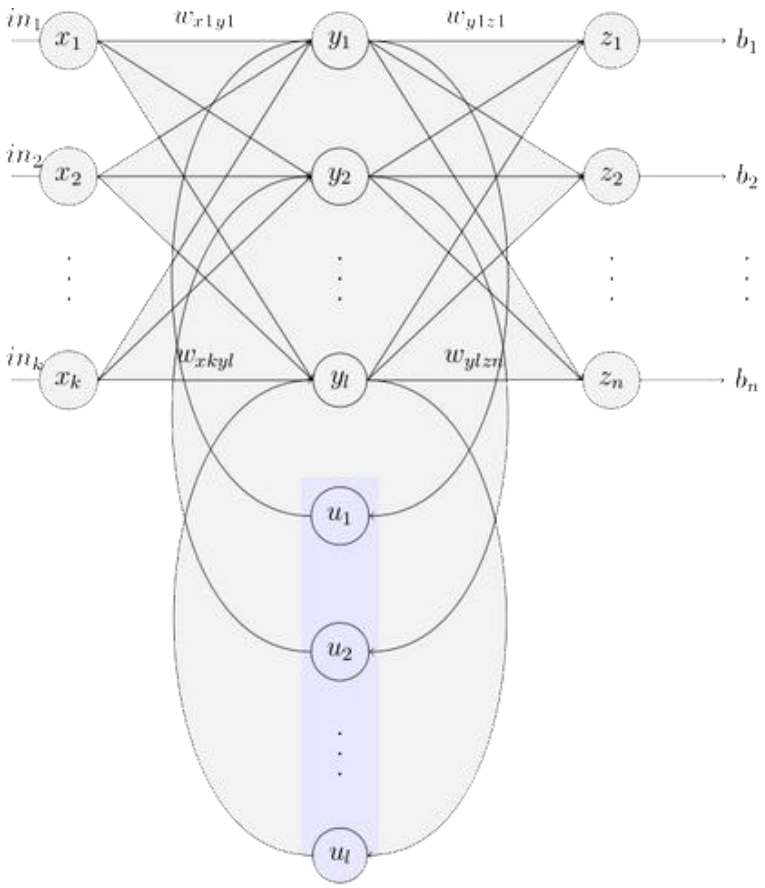


Рис. 12. Рекурентна нейронна мережа Елмана і Джодана

Важливим моментом у даній архітектурі є те, що на кожному такті вхід поширюється звичайним прямим способом, а тільки потім застосовується правило навчання [15]. Таким чином, досягається максимальна швидкість роботи нейронної мережі.

Архітектура згорткової нейронної мережі добре підходить для аналізу зображення або відео, через те, що вона ефективно шукає локальні шаблони, що зустрічаються в зображеннях чи відео, але ця архітектура не є універсальною і не підходить для всіх задач. Наприклад, вона погано підходить для задач аналізу послідовністей, до яких можна віднести автоматичне розпізнавання голосу, автоматичний переклад чи обробку текстів на рідній мові.

Часто зустрічається задача передбачення наступного слова. Слід використати аналіз послідовності при обробці загальної мови. Це потрібно для генерації текстів, а також при автоматичній обробці голосу [16]. Наприклад, ми маємо послідовність зі слів і нам треба передбачити наступне слово. Для вирішення задач такого роду використовується метод n-грам.

У нас є великий дата-сет текстів, наприклад, всі тексти з вікіпедії і ми шукаємо в них словосполучення 3 and слів з 2, 3, 4 і 5, ці словосполучення і називаються n-грамами, потім вибираємо, які n-грами зустрічаються частіше, слова з таких n-грам і підставляємо. Проблема з таким підходом полягає в тому, що потрібні нам дані можуть зустрічатися на значній відстані від один одного. Наприклад, в досить великому тексті у нас можуть зустрічатися 2 пропозиції, потім кілька інших пропозицій або навіть абзаців тексту, а потім наступна пропозиція, в якій потрібно передбачити останнє слово [17]. Але для того, щоб була можливість встановити цей зв'язок, нам необхідно пам'ятати перше речення протягом досить довгого часу, це показано на рис. 13.

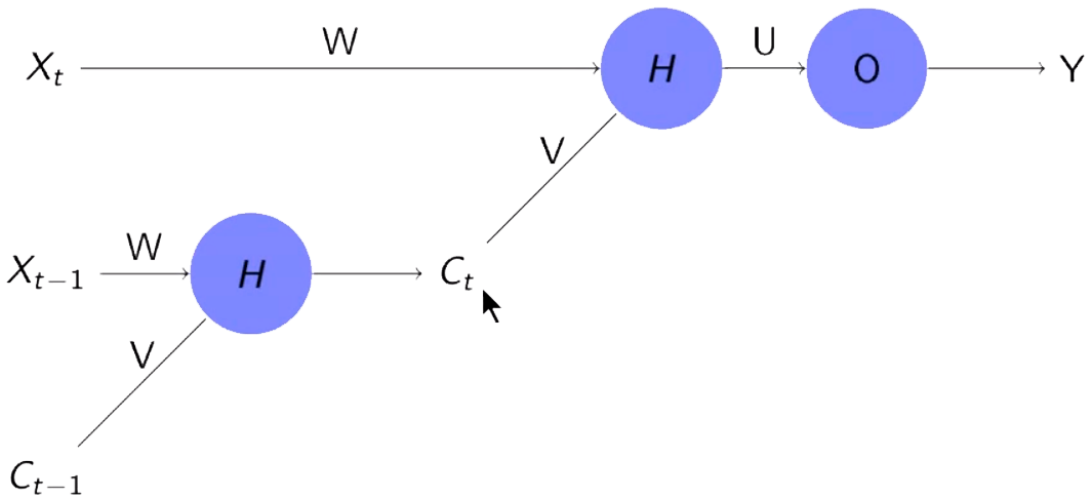


Рис. 13. Навчання нейронної мережі

Метод n-грам не дозволяє це зробити, згорткові нейронні мережі теж, як і інші нейронні мережі з прямим поширенням сигналу. А рекурентна нейронна мережа добре підходить для обробки послідовностей після того, як ми представили рекурентну нейронну мережу у вигляді мережі з прямим розповсюдженням сигналу. Для навчання такої мережі можна застосувати алгоритм зворотного поширення помилки. Ми використовуємо метод навчання з учителем і знаємо, яку правильну відповідь мережа повинна видати на останньому етапі робіт, відповідно, ми можемо порахувати помилку на останньому етапі роботи і за допомогою методу зворотного поширення помилки розрахувати, яка помилка буде на всіх попередніх шарах, тобто на копіях нашої рекурентної мережі в різні моменти часу. Зараз на практиці широко використовується спеціальна архітектура рекурентних нейронних мереж, яка називається мережа LTSM (довго короткострокова пам'ять) – це спеціальна архітектура рекурентної нейронної мережі, яка дозволяє запам'ятовувати дані на значно великий час – кілька сотень або навіть тисяч часових кроків [18]. Архітектура мережі була запропонована у 1997 році. Основний елемент цієї мережі – це спеціального виду нейрон, який використовується в якості елементу пам'яті. До нейрона надходять дані на вхід і видаються на вихід, крім цього у нейрона є зрозумілий зв'язок зі своїм ходом, вага зв'язку – це одиниця. Таким чином, якщо на вхід не надходить ніяких нових даних, то на кожному етапі значення в цьому нейроні переписується і таким чином зберігається [19]. Для управління таким нейронним осередком пам'яті використовується три вентиля:

- Вхідний вентиль.
- Вихідний вентиль.
- Вентиль забуття.

Значення цих вентилів встановлюються іншими нейронами мережі. Розглянемо, як працює такий осередок пам'яті. Припустимо, що на вхід подається перше завдання. Рекурентні мережі – це такі мережі, в яких

дозволені цикли. Нейрони в такій мережі можуть частину інформації передавати собі на вхід і за рахунок цього запам'ятовувати події, які відбулися деякий час назад і навіть великий час назад. Навчання рекурентної мережі можна представити у вигляді мереж з прямим розповсюдженням сигналу розгорнуту в часі. Мережа отримує на вхід вхідний сигнал, видає значення, а також видає ще один сигнал, який подається на вхід такої ж копії мережі тільки на наступному етапі часу [20]. На цьому етапі часу мережа отримує на вхід інший сигнал і видає на вихід наступний сигнал і так далі. Для того, щоб це значення записалося в нейрон, необхідно вхідний вентиль встановити в одиницю, після цього значення записується і зберігається в нейроні за рахунок рекурентної мережі зворотного зв'язку, це можна побачити на рис. 14.

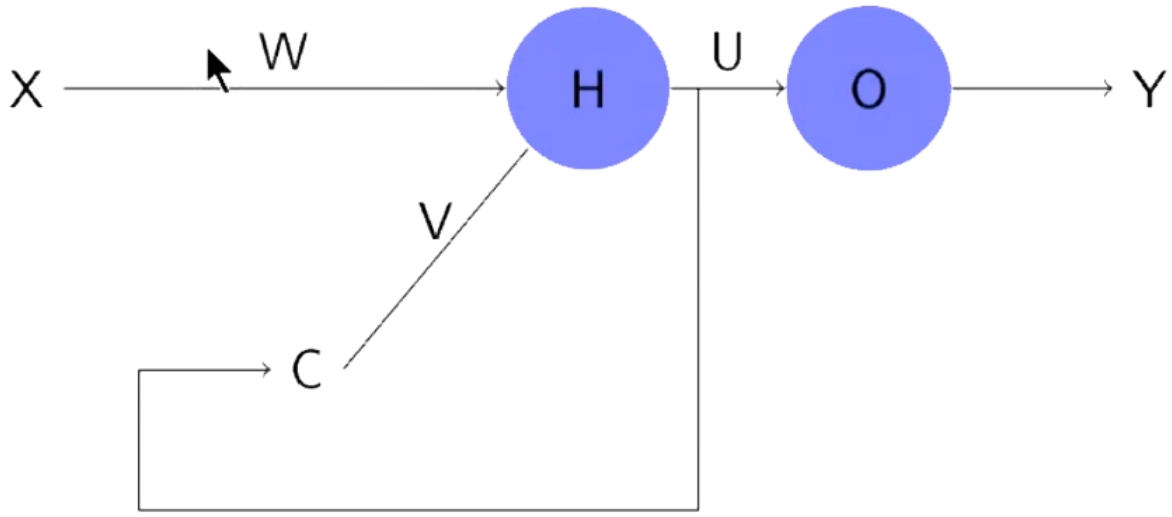


Рис. 14. Рекурентна нейронна мережа Елмана

Коли вхідний вентиль встановлений в 0, значення, які надходять на вхід нейрона, не впливають на його зміст. Якщо ми хочемо отримати на виході значення, яке було збережено в осередку, то вихідний вентиль необхідно встановити в одиницю [21]. А якщо ми хочемо забути значення, яке зберігається в осередку, то вентиль забуття необхідно встановити в нуль,

після цього значення буде стерте з нейрона і його можна використовувати для запам'ятовування іншого значення. Також в рекурентних мережах дозволений цикл, такі мережі добре підходять для обробки послідовностей. Це часто потрібно в аналізі текстів на природній мові, а також в інших областях. Для навчання рекурентних мереж використовується той же самий алгоритм зворотного поширення помилки, але перед навчанням необхідно уявити рекурентну мережу у вигляді мережі з прямим розповсюдженням сигналу шляхом розгортання в часі. Таким чином, навіть якщо в рекурентній нейронній мережі всього лише один шар, все одно в ній використовується глибоке навчання через розгортання цього шару в часі.

Нейронні мережи Елмана та Джорана також відомі, як “прості нейронні мережі”.

Нейронна мережа Елмана (1)

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y) \quad (1)$$

Нейронна мережа Джордана (2)

$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y) \text{, де} \quad (2)$$

- x_t – вхідний шар;
- h_t – прихований вектор шару;
- y_t – вихідний вектор;
- W, U та b – параметри матриці та вектору;
- σ_h та σ_y – активатори функцій.

2.2. Аргументація вибору методу машинного навчання

Для визначення можливостей для покращення методу доцільно розглянути детально базовий метод зворотного розповсюдження помилки. Сенс в тому, щоб почати використовувати навчальні вибірки, дивлячись на результат на вихідному шарі, а потім вираховувати різницю між правильною відповіддю і вихідною, щоб потім похибку поширити на попередні шари для коригування ваг. Тобто, наприклад, у нас є якась навчальна вибірка, де вказано що при деяких вхідних значеннях повинна бути визначена відповідь [22]. У нашому випадку *x* – це вхідний сигнал, для правильних відповідей випадково встановлюємо початкові ваги, проганяємо вибірку та отримуємо відповідь. Дана концепція повністю відображається самою назвою методу поширення помилки. Даний метод наведений на рис. 15.

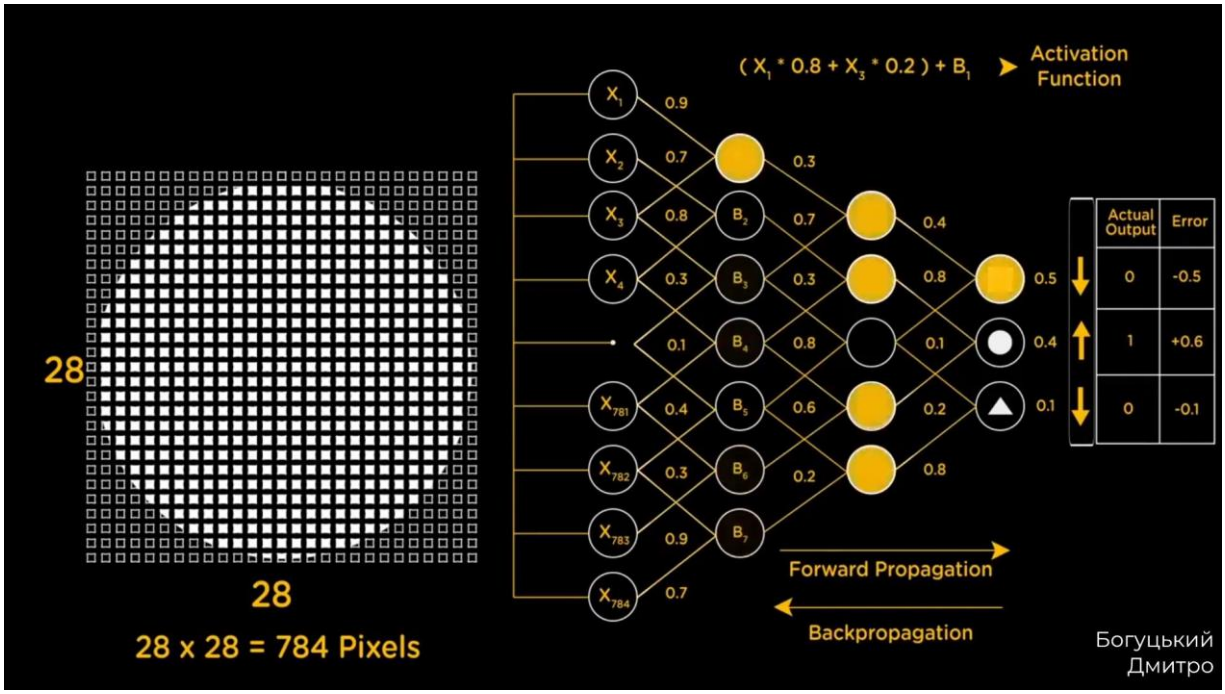


Рис. 15. Процес навчання рекурентної нейронної мережі

На вхідний шар може подаватися що завгодно, головне мати якомога більшу вибірку і проганяти від сотні до 10000 раз з постійним коригуванням ваг. Коли нейронна мережа почне стабільно видавати результати, близькі до правильної відповіді, мережу можна вважати навченою [23]. Тут знову ж можна провести паралель з людиною, яка відправилася в секцію

рукопашного бою. На початку шляху тренер просить від нього зробити одне, учень хоч і слухає уважно, але виконати завдання поки не виходить. Удар не швидкий, захоплення неправильне, тощо. Кожен раз є якась похибка в його діях, як і у нейронної мережі, а тренер з кожним тактом вказує на неї, постійно вимагає від учня правильні дії. З часом у учня з'являється так звана м'язова пам'ять, концентрація збільшується і він вже здатний більш точно відтворювати вказівки тренера, похибка зменшується і так далі. Даний вид навчання нейронної мережі називається навчання з учителем. Ми маємо відповідь на вихідні значення, тобто ми завжди можемо порівняти отримане рішення з вірною відповіддю. Але чи доцільно після кожного циклу заміряти похибки ручним чином? Змінювати ваги мережі ми можемо автоматично [24]. Адже під навчанням мережі мається на увазі наявність спеціальних програм, які будуть керувати нейроною мережею, використовуючи вибірку, і самостійно заміряти результати після кожного циклу, обчислювати похибку і коригувати ваги. Найвідомішим алгоритмом машинного навчання є алгоритм градієнтного спуску. Весь алгоритм градієнтного спуску можна проілюструвати графіком на рис. 16.

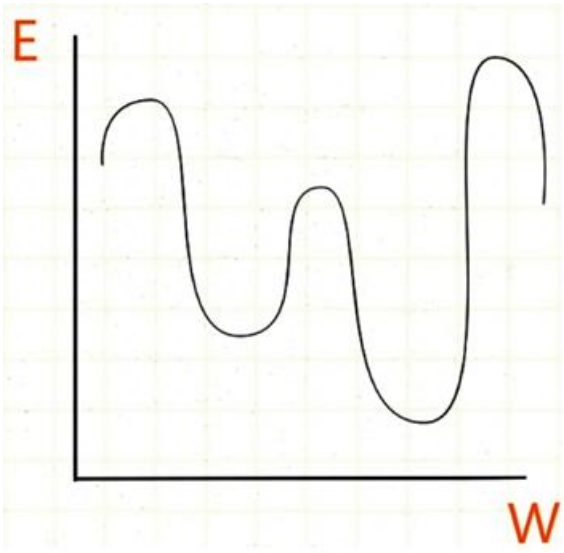


Рис. 16. Градієнтний спуск – метод машинного навчання

По осі *X* знаходиться вага нашої мережі, а по осі *Y* величина помилки і весь інтерес полягає в тому, що, підбираючи ваги, у нас виходить не якась монотонна функція, де скажімо, при першій вазі у нас помилка велика, а при наступних доборах ваг помилка зменшується [25]. Тут графік виглядає наступним чином; це так звана поверхня помилки для одного нейрона і по графіку видно, що спочатку помилка може зменшуватися, потім підбираючи ваги, далі збільшуватися, продовжуючи цю процедуру, різко зменшуватися, навіть значніше, ніж раніше. Скочуючись або стрибаючи з пагорбів, потрібно знайти абсолютний мінімум функції, тобто вагу, при якій помилка буде найменшою. В даному випадку це друга западина і вся небезпека в тому, щоб не потрапити в локальні мінімуми функції. Тобто западини, які є мінімальними тільки на певній ділянці функції, зображені на рис. 16. Для простоти сприйняття ми можемо уявити кенгуру, який стрибками пересувається з пагорба до западини, величина стрибка задається параметром швидкості навчання і на перший погляд може здатися, що чим вище швидкість, тим швидше ми навчимо мережу [26]. Але це не так. Коли людина читає підручник з історії, вона займається цим поверхнево. Вона читає по одному слову декілька сторінок і для того, щоб зрозуміти прочитане, треба на деякий час зупинитися. Так і в навчанні нейронної мережі, при дуже великій швидкості навчання кенгуру просто перестрибне потрібну нам вагу, так до того ще й побачить, що далі тільки можна рухатись в гору до помилок, і буде намагатися стрибнути назад, де помилка була менше змінюватися. Але ми не можемо встановити надто малу швидкість, оскільки це буде занадто довгим процесом. Потрібно знайти «золоту середину». Тут ще на допомогу приходить інший параметр це момент, він повинен допомогти знизити швидкість навчання при наближенні до западини і порівняння. Якщо ми зі звичайною швидкістю читаємо підручник та знаходимо важливий момент, ми можемо сповільниться, щоб обміркувати прочитане [27]. Так і тут, в зворотних випадках параметр моменту додасть прискорення для подолання локального мінімуму.

Налаштовуючи швидкість навчання і момент, використовуючи навчальну вибірку, програма коректує вагу, щоб нейронна мережа видавала потрібні нам відповіді, при тому перенавчання теж буде погано працювати. Якщо довго навчати мережу по одній вибірці, при цьому не змінюючи порядок вхідних даних з вибірки, то вона вивчить всі відповіді і запам'ятає, що наприклад, автомобіль це обов'язково 5 дверей, червоного кольору седан, а коли покажемо їй синій кабріолет, то вона не зрозуміє цього [28]. Це загальна концепція методу зворотного поширення помилки методів машинного навчання. Насправді вони більше залежать від архітектури самої мережі. Наш метод відноситься до категорії навчання з учителем. Коли у нас немає вчителя, ми подаємо різні сигнали на нашу мережу, єдине, що вона зможе зробити – це розбити ці сигнали на групу, тобто з часом її нейрони почнуть по-різному реагувати на інформацію, хтось стане спеціалістом у визначенні автомобілів, хтось пішоходів, а хтось велосипедистів. Навчання без учителя не означає, що тут немає ніякого втручання людини, може повторно знадобитися коригування як архітектури, так і самої ваги.

2.3. Особливості запропонованого методу

Після детального аналізу існуючих методів розпізнавання зображень, методів машинного навчання та методів генерації програмного коду, було вирішено створити власний метод генерації програмного коду за допомогою нейронної мережі на базі вхідного зображення.

Алгоритм представляє собою поєднання генеративно-змагальної нейронної мережі та рекурентної нейронної мережі. Як описано вище, генеративно-змагальна нейронна мережа може бути використана для розпізнавання зображень різної складності [29]. Рекурентна нейронна мережа у власному методі виконує функцію другої нейронної мережі, що навчається на результаті роботи генеративно-змагальної нейронної мережі. На рис. 17 наведено базову схему роботи власного методу.

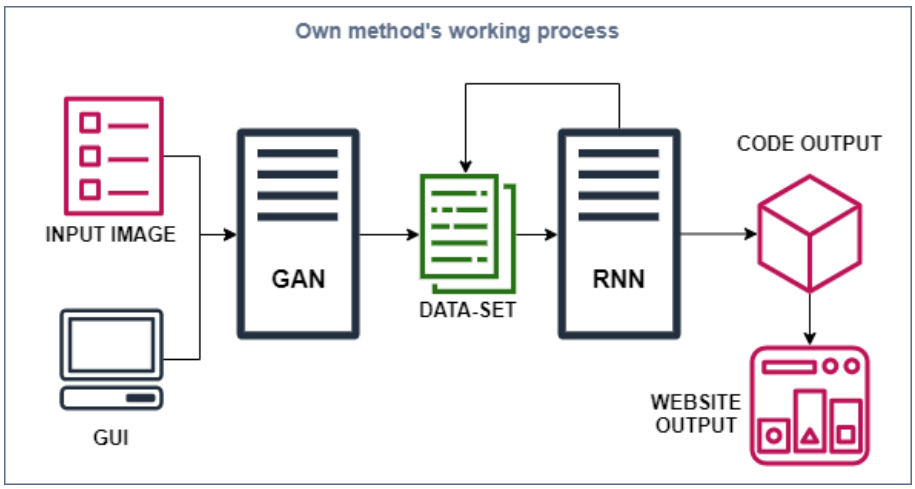


Рис. 17. Схема роботи запропонованого методу

Як видно з рис. 17, на вхід надається створене користувачем вхідне зображення, генеративно-змагальна нейронна мережа розпізнає зображення та утворює дата-сет. Цей дата-сет є вхідними даними для другої рекурентної нейронної мережі, яка на базі дата-сету генерує програмний код, з якого на фінальному етапі роботи утворюється веб-сторінка, яку прагнув створити користувач. З особливостей власного методу можна виділити наступне:

- Зменшення часу обробки зображення.
- Зменшення часу обробки дата-сету.
- Загальне прискорення алгоритму.
- На виході генерується адаптивний програмний код.

Більш детально про роботу власного алгоритму написано у третьому розділі.

2.4. Висновки до розділу 2

В даному розділі було проведено грунтовну роботу з дослідження існуючих методів глибинного навчання, проаналізовано існуючі типи нейронних мереж. Відокремлено недоліки та переваги існуючих алгоритмів. Було виконане наступне:

- Вибір методу розпізнавання зображення.

- Вибір методу розпізнавання тексту.
- Вибір алгоритм машинного навчання.
- Опис власного методу.

Запропоновано створення власного програмного методу генерації програмного коду за допомогою GAN та RNN на базі вхідного зображення, в якому покращено або виправлено недоліки існуючих аналогів. Детально запропонований метод описаний в третьому розділі.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДУ

3.1 Засоби розроблення програмного забезпечення

З огляду на те, що програмний метод складається з користувацького інтерфейсу та серверної частини, яка створює веб-сторінки за допомогою мов HTM та CSS, було вирішено реалізовувати програмний продукт у вигляді веб-застосунку, що з’єднаний з серверною частиною, а саме нейронними мережами, що працюють на віддаленому сервері. Програмний продукт складаєтсья з таких компонентів:

- Користувацький інтерфейс:
    - Можливість реєстрації та авторизації користувача.
    - Можливість додавання вхідного зображення.
    - Можливість налаштування роботи застосунку.
    - Робота застосунку без перезавантаження сторінки.
- Серверна частина:
    - З'єднання застосунку з віддаленим сервером.
    - Генерація вихідного програмного коду.
    - Налаштування локального серверу.
    - Прописування залежностей у вихідному проекті.

Розглянемо найпопулярніші інструменти для створення користувацьких інтерфейсів.

3.1.1. *ReactJS*

Java Script знайшов велику популярність останнім часом і стала однією з найпопулярніших мов програмування в світі. Сьогодні налічується сотні різних фреймворків і бібліотек та наборів різних технологій. Феріймворк – це інструмент для побудови динамічних веб-застосунків або настільних додатків на мові Java Script [30]. Як і з будь-яким іншим інструментом, розробники вдаються до використання фреймворків там, де неможливо, або принаймні дуже складно і дуже довго виконати завдання звичайними засобами. В переважній більшості випадків фреймворк використовується для написання так званих SPA (Single Page Application). Тобто все, що відбувається на сайті, відбувається на одній сторінці без прямого переходу з неї. За допомогою фреймворків можна розробляти як повноцінний сайт, так і функціональні модулі – так звані онлайн-інструменти. Звичайно, повноцінні фреймворки краще підходять для першого завдання, а для другого рекомендується використовувати простіші фреймворки або бібліотеки. Але спочатку слід трохи проаналізувати роботу фреймворків. Вони надають чітку структуру програми та реалізуються з використанням патернів проектування. Це поняття більше використовується для серверної частини, але явно має місце для розроблення клієнтської частини з використанням фреймворків Java Script. Найбільш широко поширені такі патерни, як MVC (Model View Controller) – це модель, вид та контролер [31]. MVP (Model View Presenter) – модель, вид та представник, MVVM (Model–view–viewmodel). Найпопулярнішою є модель вид-контролер. З цього патерну побудовано багато фреймворків. Нові фреймворки дуже зручні для побудови односторінкових сайтів, які повинні бути насичені функціональними можливостями. Також використання фреймворку зобов'язує програмістів мати строгу структуру

компонентів. Коду стає помітно менше і він «чистіший», що позитивно відбивається на швидкості розроблення, також підтримка коду є простішою. Наявність структури, мається на увазі модульність додатків, дозволяє працювати над додатком кільком розробникам одночасно. Інша перевага більше випливає на використання самого Java Script, але значно посилюється при використанні фреймворків, це можливість швидко створювати мобільні чи настільні кросплатформені додатки. З істотних недоліків можна виділити неповну підтримку пошуковими системами, але це завдання рідко збігається з завданням з реалізації SPA, тим більше, що провідні пошукові системи вже практично повністю вирішили цю проблему. Додатків на фреймворках вже випущено дуже багато і цей сегмент тільки набирає популярності. Головна перевага фреймворків – це оновлення сторінки без перезавантаження. При виведенні сторінки на сервері буде генеруватися вся розмітка. Якщо користувач захоче отримати ще дані, йому доведеться оновити всю сторінку, якщо користувач виконає певну дію на сторінці, знову доведеться оновити всю сторінку. React дозволяє створювати динамічні та інтерактивні інтерфейси. В даному випадку клієнтська та серверна частини програмного забезпечення є незалежними одиницями, які можуть спілкуватися. Не потрібно робити оновлення всієї сторінки на кожен клік, на сервері програміст працює тільки з даними і не генерує розмітку, що позитивно позначається на продуктивності сервера. JQuery була лише бібліотекою, на її зміну прийшли повноцінні фреймворки. ReactJS вирішує проблему роботи з користувачем і дозволяє створювати повноцінну серверну функціональність на сайті, та не вимагає перезавантаження сторінки. Це не тільки зручно, але й дуже привабливо як для користувача, так і для розробника.

Далі наведено приклад коду ReactJS на мові JavaScript.

Лістинг 1. Приклад коду фреймворку ReactJS на мові JavaScript

```
import Carousel from 'react-multi-carousel';

import './WithScrollbar.css';

class WithScrollbar extends React.Component {
  state = { additionalTransfrom: 0 };
  render() {
    const CustomSlider = ({ carouselState }) => {
      let value = 0;
      let carouselItemWidth = 0;
      if (this.Carousel) {
        carouselItemWidth = this.Carousel.state.itemWidth;
        const maxTranslateX = Math.round(
          // so that we don't over-slide
          carouselItemWidth * (this.Carousel.state.totalItems - this.Carousel.state.slidesToShow) +
            150
        );      const { transform } = carouselState;
      return (
        <div className="custom-slider">
          <input
            type="range"
            value={Math.round(Math.abs(transform) / value)}
            max={
              (carouselItemWidth * (carouselState.totalItems - carouselState.slidesToShow) +
                (this.state.additionalTransfrom === 150 ? 0 : 150)) /
              value
            }
            onChange={e => {
              if (this.Carousel.isAnimationAllowed) {
                this.Carousel.isAnimationAllowed = false;
              }
              const nextTransform = e.target.value * value;
              const nextSlide = Math.round(nextTransform / carouselItemWidth);
              if (e.target.value == 0 && this.state.additionalTransfrom === 150) {
                this.Carousel.isAnimationAllowed = true;
                this.setState({ additionalTransfrom: 0 });
              }

export default WithScrollbar;
```

3.1.2. Keras як бібліотека нейронного навчання

Keras – API нейронних мереж високого рівня, написаний на Python і здатний працювати над TensorFlow, CNTK або Theano. Keras був розроблений з акцентом на швидке експериментування [32]. Вміння переходити від ідеї до результату з найменшою можливою затримкою є ключовим для хорошого дослідження.

Було вирішено використати Keras, адже ця бібліотека глибокого навчання, яка:

- Дозволяє легко та швидко прототипувати (завдяки зручності користування, модульності та розширюваності).
- Підтримує як конволюційні мережі, так і періодичні мережі, а також їх комбінації.
- Працює безперебійно на процесорі та GPU.

Також слід розглянути керівні принципи Keras, які підходять для написання власного програмного продукту.

- **Зручність у користуванні.** Keras – це API, призначений для людей, а не для машин. Keras дотримується кращих практик зменшення когнітивного навантаження: він пропонує послідовні та прості API, мінімізує кількість дій користувачів, необхідних для випадків загального використання та забезпечує чіткий та дієвий зворотний зв'язок при помилці користувача.
- **Модульність.** Модель розуміється як послідовність або графік самостійних, повністю налаштовуваних модулів, які можна підключити разом з якомога меншими обмеженнями. Зокрема, нейронні шари, функції витрат, оптимізатори, схеми ініціалізації, функції активації та схеми регуляризації – це окремі модулі, які можна комбінувати для створення нових моделей.
- **Легка розширюваність** Нові модулі легко додавати (як нові класи та функції), а існуючі модулі містять достатньо прикладів. Можливість легко створювати нові модулі дозволяє

отримати повну виразність, що робить Keras придатним для розширених досліджень.

- **Робота з Python**. Немає окремих файлів конфігурації моделей у декларативному форматі. Моделі, описані в коді Python, який компактний, простіший у налагодженні та легко масштабується.

Лістинг 2. Приклад коду з бібліотеки Keras

```
train_df = pandas.read_csv("./train.csv")
valid_df = pandas.read_csv("./valid.csv")

train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_dataframe(
        dataframe=train_df,
        directory='data/train',
        x_col="filename",
        y_col="class",
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

validation_generator = test_datagen.flow_from_dataframe(
        dataframe=valid_df,
        directory='data/validation',
        x_col="filename",
        y_col="class",
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

model.fit_generator(
        train_generator,
        steps_per_epoch=2000,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=800)
```

Apply_transform(x, transform_parameters) застосовує перетворення до зображення відповідно до заданих параметрів [33].

Аргументи:

- x : 3D тензор, одиночне зображення.
- transform_parameters: Словник із парами рядків – параметри, що описують перетворення. В даний час використовуються наступні параметри зі словника:
    - 'theta': float. Кут повороту в градусах.
    - 'tx': float. Зсув у напрямку x.
    - 'ty': float. Зсув у напрямку y.

- 'shear': float. Кут зсуву в градусах.
- 'zx': float. Масштаб у напрямку x.
- 'zy': float. Масштаб у напрямку y.
- 'flip_horizontal': boolean. Горизонтальний фліп.
- 'flip_vertical': boolean. Вертикальний фліп.
- 'channel_shift_intencity': float. Інтенсивність зсуву каналу.
- 'brightness': float. Інтенсивність зсуву яскравості.

Flow(x, y=None, batch_size=32, shuffle=True, sample_weight=None, seed=None, save_to_dir=None, save_prefix='', save_format='png', subset=None) знімає масиви даних і міток, генерує партії доповнених даних.

Аргументи:

- x : Вхідні дані. Numpy масив рангу 4 або кортеж. Якщо кортеж, перший елемент повинен містити зображення, а другий елемент – інший великий масив або список масивів, який передається на вихід без будь-яких змін. Може використовуватися для подачі різних даних моделі разом із зображеннями. У випадку даних масштабу сірого, вісь каналів масиву зображень повинна мати значення 1, у випадку даних RGB вона повинна мати значення 3, а у випадку даних RGBA – значення 4.
- y : Мітки
- batch_size : Int (за замовчуванням: 32).
- перетасування : логічне значення (за замовчуванням: True).
- зразковий вага : зразки ваги.
- насіння : Int (за замовчуванням: None).
- save_to_dir : None або str (за замовчуванням: None). Це дозволяє необов'язково вказати каталог, в який зберігати створені збільшені зображення (корисно для виконання візуалізації процесу).

- save_prefix : Str (за замовчуванням :)" . Префікс, який потрібно використовувати для назв збережених зображень (іменовано лише те, якщо save_to_dir встановлено).
- save_format : один із "png", "jpeg" (стосується лише, якщо save_to_dir встановлено). За замовчуванням: "png".
- підмножина : Підмножина даних ("training"або "validation"), якщо validation_split встановлено ImageDataGenerator.

Розглянемо наступні обмеження даної бібліотеки.

Функції constraints модуля дозволяють встановлювати обмеження (наприклад, негативні штрафи) застосовуються на рівні шару. API буде залежати від шару, але шари Dense, Conv1D, Conv2Di Conv3D мають єдину API.

Ці шари розкривають 2 аргументи ключових слів:

- kernel_constraint для матриці основних ваг;
- bias_constraint для упередженості.

Лістинг 3. Приклад підключення бібліотеки через TensorFlow

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

3.2. Архітектура розробленого програмного забезпечення

З огляду на можливі області застосування даного алгоритму, було вирішено, що архітектура системи повинна бути побудована за клієнт-серверною моделлю модульного типу. Так як для реалізації користувацького інтерфейсу був обраний Java Script фреймворк ReactJS, за основу була взята модульна, компонентна архітектура. Для реалізації

серверної частини програмного продукту за основу була взята бібліотека Keras, яка написана на мові програмування Python, генеративно-змагальна нейронна мережа та рекурентна нейронна мережа.

3.2.1. *Серверна архітектура програмного забезпечення*

В основі серверної архітектури лежать дві нерйонні мережі. Перша нейронна мережа GAN (Generative adversarial network) – генеративно-змагальна нейронна мережа, що добре підходить для розпізнавання зображень. Друга нейронна мережа – це рекурентна нейронна мережа, задачею якої є генерація програмного коду веб-сторінки на базі дата-сету, який був утворений в результаті роботи GAN. На рисунку 18 можна побачити основну серверну архітектуру запропонованого програмного забезпечення.

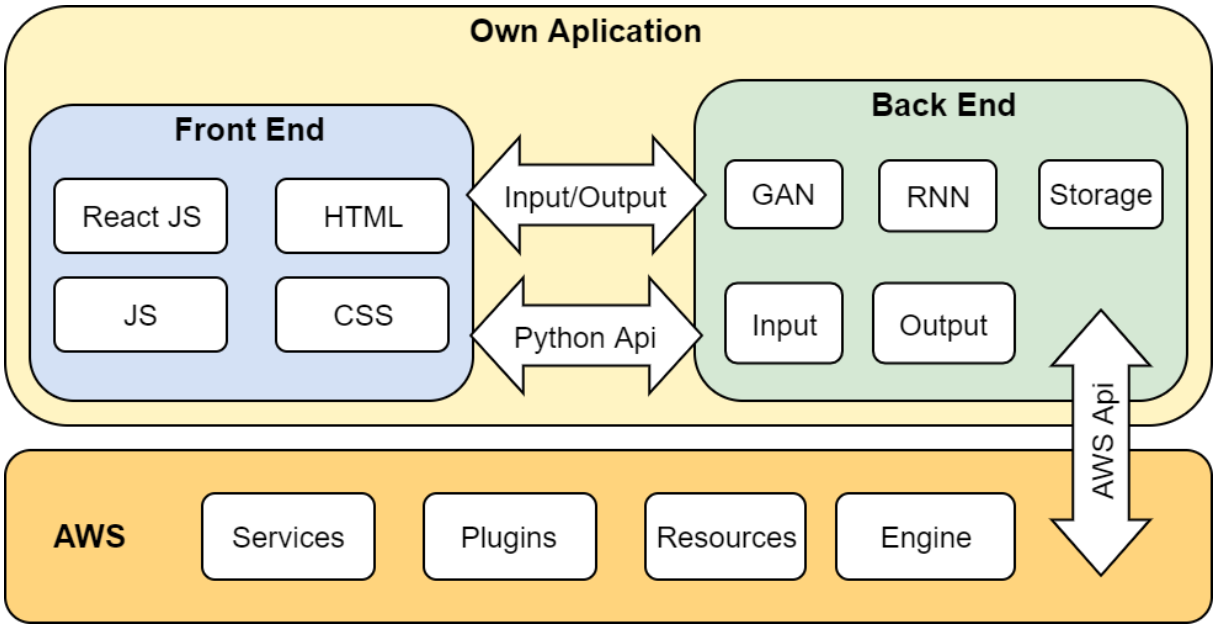


Рис. 18. Архітектура програмного забезпечення

Як показано на рисунку 18, вхідними даними для генеративно-змагальної нейронної мережі є вхідне зображення, яке користувач завантажив до GUI програмного забезпечення. Вхідне зображення може бути будь-якого формату, притаманного до загальних форматів зображень,

таких, як PNG або JPEG. Далі генеративно-змагальна нейронна мережа аналізує вхідне зображення і утворює так-зване дерево розбіру, яке написане на DSL мові. Це перше, що відрізняє власний метод генерації коду від існуючих аналогів та дає змогу власному методу працювати на 10% швидше від конкурентів в сфері генерації коду. Більш детально про цю та інші особливості власного методу описано у третьому пункті третього розділу.

Після того, як вхідне зображення було проаналізоване за допомогою GAN і утворений перший дата-сет, він надходить до RNN в якості вхідних даних [34]. Далі RNN на основі вхідного дата-сету генерує програмний код майбутньої веб-сторінки. На фінальному етапі RNN декілька разів проходить по згенерованому програмному коді для створення адаптивної та крос-браузерної верстки. Також на клієнтській стороні програмного забезпечення використовується локальний сервер, зазвичай на порті 4001, та відбувається завантаження згенерованого програмного коду на нього. На даному етапі робота програмного забезпечення закінчена, про що користовачеві повідомляє користувацький інтерфейс. Далі слід детальніше розглянути особливості власного методу генерації коду та створення власного дата-сету.

3.3. Особливості створення дата-сету

На відміну від існуючих рішень, у власному методі створення дата-сету реалізовано за допомогою створення дерева, що являє собою алгоритм, написаний на DSL мові. DSL (domain-specific language) – це комп’ютерна мова, що спеціалізується на певній предметній області. Розглянемо процес створення такого дерева більш детально. Наприклад, користувач завантажив вхідне зображення, як проілюстровано на рис. 19.

Рис. 19. Вхідне зображення користувача №1

Після завантаження відного зображення в користувацький інтерфейс воно надходить до генеративно-змагальної нейронної мережі, яка повинна утворити з нього дата-сет з використанням DSL мови. Завдяки створенню власного дата-сету на базі DSL мови за допомогою генеративно-змагальної нейронної мережі вдалося досягти першого прискорення роботи алгоритму. Розглянемо процес створення дата-сету більш детально. Після розпізнавання простого вхідного зображення, наведеного на рис. 19, генеративно-змагальна нейронна мережа генерує дата-сет, який проілюстровано на рис. 20.

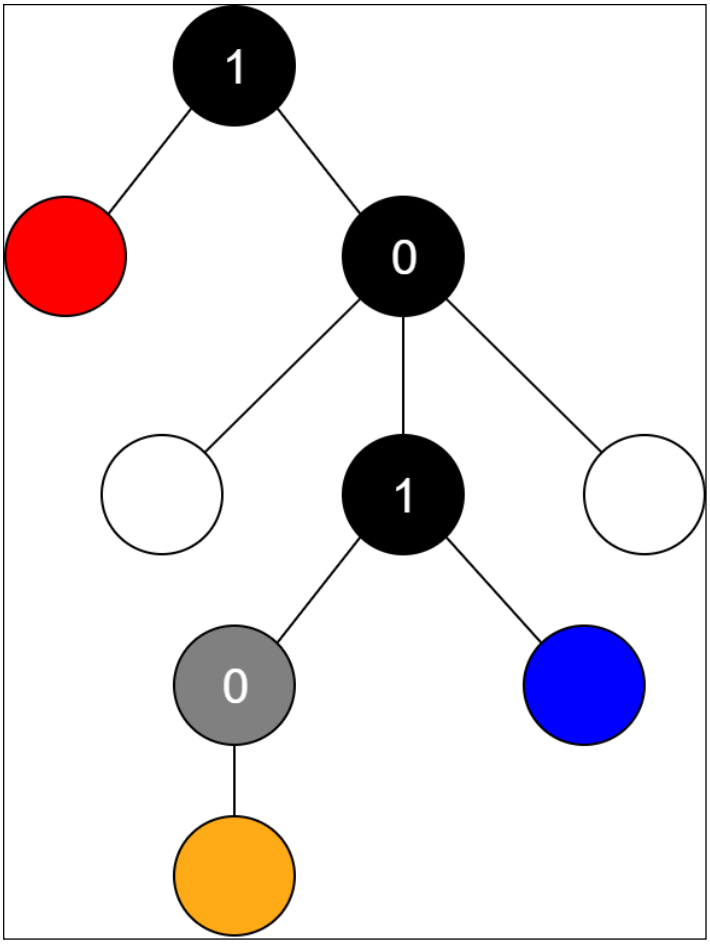


Рис. 20. Утворене DSL дерево на базі вхідного зображення №1

Для створення власного дата-сету було вирішено створити свою сітку розбіру. Таким чином, впроваджено декілька правил для розпізнавання логічних HTML елементів, зображених на вхідному зображені. Чорними кружками позначені логічні вузли: 1 – рядок, 2 – стовпець. За цими правилами відбувається розбір зображення. Ми можемо прослідкувати, чи правильно працює алгоритм. На рис. 19 ми бачимо червоний header (елемент вгорі веб-сторінки), біле body (тіло сайту), сірий container (область, що обмежує тіло сайту), синій footer (нижня частина веб-сторінки) та жовтий sidebar (бокове меню веб-сторінки). Розглянемо більш складний випадок. Наприклад, маємо вхідне зображення, зображене на рис. 21.

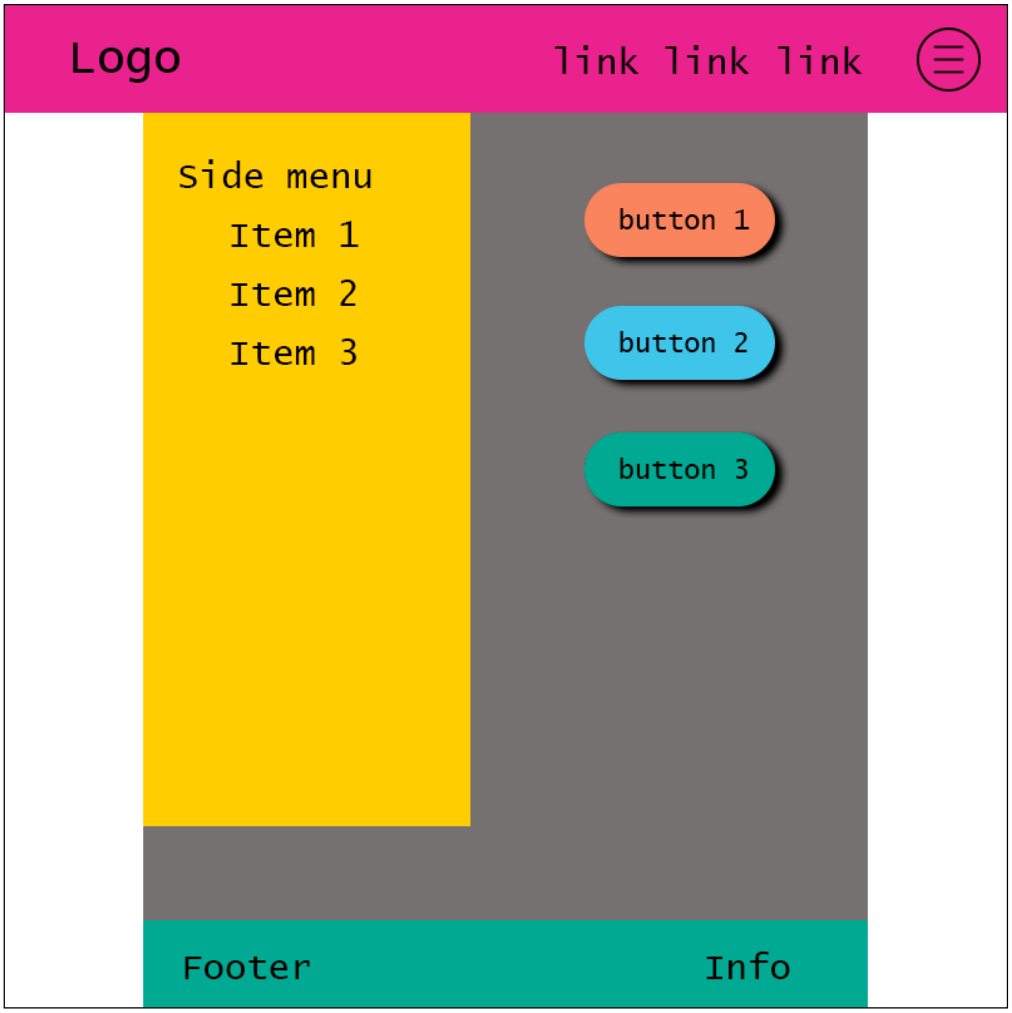


Рис. 21. Вхідне зображення користувача №2

Вхідне зображення, наведене вище, вже є більш складним з точки зору розпізнавання зображення нейронною мережею. Але завдяки тому, що генеративно-змагальна нейронна мережа була навчена на базі 10000 веб-сторінок з відкритого доступу, розпізнавання цього вхідного зображення не є для неї проблемою. Таким чином, використовуючи вище наведені правила розпізнавання HTML блоків, генеративно-змагальна нейронна мережа побудувала наступний DSL дата-сет, наведений на рис. 22.

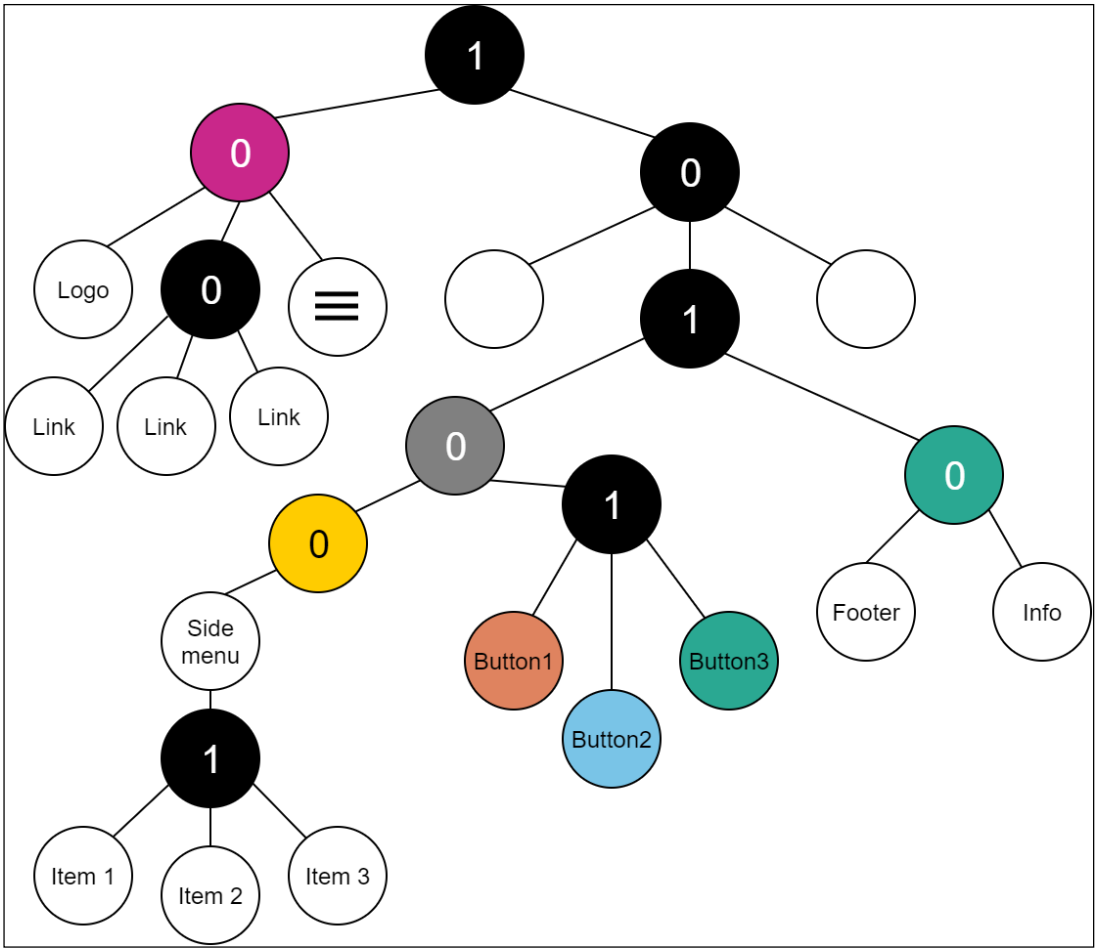


Рис. 22. Утворене DSL дерево на базі вхідного зображення №2

На цьому, вже більш складному дереві, видно багато різних вузлів, таких, як:

- Header:
  - Logo.
  - Navbar.
  - Burger menu.
- Container:
  - Sidebar:
    - Side menu:
      - Item 1.
      - item 2.
      - Item 3.

    - Button 1.
    - Button 2.
    - Button 3.
- Footer:
    - Footer info.

Наступним етапом роботи власного методу є передача цих DSL дерев до другої рекурентної нейронної мережі для подальшої генерації програмного коду. Розглянемо більш детально цей процес у наступному розділі.

3.4 Особливості генерації коду

На даному етапі роботи власного алгоритму відбувається генерація програмного коду за допомогою рекурентної мережі на базі вхідного дата-сету від генеративно-змагальної нейронної мережі. Нейронна мережа навчалася на дата-сетах з 10000 веб-сторінкок, і ми можемо оцінювати продуктивність мережі на одному і тому ж наборі даних [35]. Це лише дасть нам уявлення про те, наскільки добре ми моделювали набір даних (наприклад, точність блоків HTML), але ми не маємо уявлення про те, наскільки добре алгоритм може працювати на нових даних. Це зроблено для простоти, але ми можемо розділити дата-сет на набори блоків для навчання та оцінки нашої моделі.

Ми можемо оцінити нашу модель на навчальному наборі даних за допомогою функції evaluate() на нашій моделі та передати їй ті самі вхідні та вихідні дані, які використовувались для тренування моделі.

Ця функція генерує прогнозування для кожної пари вводу та виводу та збирає бали, включаючи середню втрату та будь-які налаштовані користувачем показники, такі як точність.

Функція evaluate() поверне список з двома значеннями. Перше – це втрата моделі на наборі даних, а друге – точність моделі на наборі даних.

Нас цікавить лише повідомлення про точність, тому ми будемо ігнорувати значення втрат.

Лістинг 4. Приклад коду генерації коду за допомогою Keras

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

Запустивши цей код, ми повинні побачити повідомлення для кожної з 150 епох, що друкують втрати та точність, з подальшою остаточною оцінкою навченої моделі на навчальному наборі даних.

На віддаленому сервері, що працює на двох відеокартах GTX 1080, потрібно близько 10 секунд [36].

Ідеальною ситуацією є наближення втрат до нуля, а точність – до 1,0 (100%). Це неможливо для будь-яких, але найбільш тривіальних проблем машинного навчання. Натомість у нашій моделі завжди буде помилка. Мета полягає у виборі конфігурації моделі та конфігурації тренувань, які

дозволять досягти найменших втрат та максимально можливої точності для заданого набору даних.

Розглянемо приклад згенерованого коду до вхідного зображення №2, що проілюстрований в лістингу 5.

Лістинг 5. Приклад згенерованого коду по зображеню №2

```
<body>
  <section class="header">
   <div class="container">
    <div class="logo">
     <img src="assets/img/logo.png" alt="logo" />
    </div>
    <ul class="navbar">
     <li class="navbar__item">
      <a href="#" class="navbar__link">link</a>
     </li>
     <li class="navbar__item">
      <a href="#" class="navbar__link">link</a>
     </li>
     <li class="navbar__item">
      <a href="#" class="navbar__link">link</a>
     </li>
    </ul>
    <div class="burger-menu">
     <img src="assets/img/menu.png" alt="menu" />
    </div>
   </div>
  </section>
  <section class="main">
   <div class="main-container">
    <div class="sidebar">
     <h2 class="sidebar__title">Side menu</h2>
     <ul class="sidemenu">
      <li class="sidemenu__item">
       <a href="" class="sidemenu__link">
        Item 1
       </a>
      </li>
      <li class="sidemenu__item">
       <a href="" class="sidemenu__link">
        Item 2
```

```
          </a>
         </li>
         <li class="sidemenu__item">
          <a href="" class="sidemenu__link">
           Item 3
          </a>
         </li>
        </ul>
       </div>
       <div class="button_container">
        <button class="btn btn-orange">button 1</button>
        <button class="btn btn-blue">button 2</button>
        <button class="btn btn-green">button 3</button>
       </div>
      </div>
    </section>
    <section class="footer">
     <div class="container">
      <h2 class="footer__caption">Footer</h2>
      <h2 class="footer__info">Info</h2>
     </div>
    </section>
  </body>
```

3.4.1. *Впровадження модифікацій*

Після впровадження вище зазначених методів спостерігаються доволі великі прирости в продуктивності програмного забезпечення. Детальніше це описано в розділі 4.

3.5. Особливості програмної реалізації запропонованого методу

Розроблене програмне забезпечення складається з клієнтської та серверної частини [37]. Клієнтська частина реалізована за допомогою фреймворку ReactJS на базі мови програмування Java Script. Це дозволило проводити обробку введеної користувачем інформації без перезавантаження сторінки веб-застосунку.

Серверна частина розроблена на мові програмування Python з використанням бібліотеки машинного навчання Keras [38], генеративно-

змагальної та рекурентної нейронних мереж. Також для взаємодії двох вище зазначених нейронних мереж, був впроваджений власний метод створення дата-сету з використанням DSL-мови.

Розроблене програмне забезпечення дозволяє створювати готові веб-сторінки гляхом генерації програмного коду за допомогою нейронних мереж, завантажуючи зображення шаблону сайту до користувацького інтерфейсу.

В ході розроблення програмного методу необхідно було вирішити проблему одночасної роботи двох нейронних мереж, тому для цього було вирішено створити власну DSL-мову, яка беспосередньо забезпечує цей процес [39]. Користовачу залишається лише завантажити зображення в форматі JPEG або PNG до користувацького інтерфейсу та натиснути клавішу “згенерувати”.

- Програмний додаток дає змогу створювати веб-сторінки натисканням однієї клавіші.
- Гнучке налаштування процесу.
- Запам'ятовування останніх веб-сторінок.
- Реєстрація та авторизація.

3.6. Висновки до розділу 3

У рамках дослідження можливостей модифікації алгоритмів генерації програмного коду, запропоновано метод генерації програмного коду за допомогою генеративно-змагальної та рекурентної нейронних мереж на базі вхідного зображення.

Розроблений програмний продукт містить алгоритм генерації коду з графічним інтерфейсом користувача.

Проаналізовано методи створення програмного коду на основі вихідного зображення за допомогою нейронної мережі, а саме GAN та RNN. Також проаналізовано шляхи вдосконалення існуючих методів та

запропоновано власний метод. Завдяки використанню власного методу генерації дерев DSL вдалося змусити GAN та RNN працювати разом, тим самим досягнувши 10% підвищення продуктивності. Завдяки використанню власної сітки GRID вдалося досягти адаптивності згенерованого коду. Створено користувацький інтерфейс та налаштування роботи алгоритму.

Запропонований метод не тільки має більш високу продуктивність, але і генерує код для різних браузерів і є адаптивним, якого немає в жодному з аналогів. Крос-браузерність досягається шляхом генерування попередньо встановлених властивостей постачальників, які є механізмами візуалізації сучасних браузерів. Це дозволяє налаштувати властивості для кожного окремого механізму візуалізації, щоб уникнути невідповідностей між реалізаціями.

Основними недоліками запропонованого способу є неможливість розпізнати зображення з вхідного файлу, який знаходиться в каталозі проекту і вимагає потужного віддаленого сервера для виконання необхідного методу обчислення. Натомість зображення замінюється так званим заповнювачем.

4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Методика оцінювання ефективності методів генерації коду

Для оцінки якості роботи методу генерації програмного коду необхідно дослідити та визначити ключові параметри, що є важливими для роботи самого алгоритму як в мінімальному режимі, так і в максимальному.

Після детального вивчення питання можна ввести наступні показники:

- Загальний час роботи алгоритму.
- Час роботи алгоритму на базі вхідного зображення, що складаєтсья лише з тексту.
- Час роботи алгоритму на базі вхідного зображення, що складаєтсья з тексту та зображень.
- Точність генерації коду.

Дані характеристики максимально повно оцінюють покращення в рамках генерації програмного коду.

4.2. Результати роботи запропонованого методу

Для тестування власного методу був обраний метод ручного тестування.

Для вимірювання точності генерації програмного коду виконано тестовий запуск програмного забезпечення, результати якого зображені на рис. 23.

```
...
768/768 [==============================] - 0s 63us/step - loss: 0.4817 - acc: 0.77
Epoch 147/150
768/768 [==============================] - 0s 63us/step - loss: 0.4764 - acc: 0.77
Epoch 148/150
768/768 [==============================] - 0s 63us/step - loss: 0.4737 - acc: 0.76
Epoch 149/150
768/768 [==============================] - 0s 64us/step - loss: 0.4730 - acc: 0.77
Epoch 150/150
768/768 [==============================] - 0s 63us/step - loss: 0.4754 - acc: 0.77
768/768 [==============================] - 0s 38us/step
Accuracy: 76.56
```

Рис. 23. Результати точності генерації коду на 150 епохах

У таблиці 1 відображено результати тестування точності генерації програмного коду при проходжені 150 епох. З наведеного рисунку можна побачити, що точність генерації коду в середньому становить 78 %. Далі наведено порівняльні дані точності вимірювання існуючих аналогів.

Таблиця 1

Порівняння точності генерації коду

Pix2Code	DeepCoder	Власний метод
75%	70%	78%

Загальний результат є середнім всіх спроб генерації програмного коду на основі 150 епох роботи алгоритму.

Далі слід протестувати основний показник роботи алгоритму – це загальна швидкість роботи.

Для того, щоб доцільніше протестувати швидкість роботи програмного методу, було вирішено проводити тести у двох навантаженнях:

- Вхідне зображення з текстом.
- Вхідне зображення з текстом та картинками.

Тестування загальної швидкодії програмного методу проводилось за допомогою unit та manual тестів. Результати тестів наведено в табл. 2.

Таблиця 2

Загальний час роботи методу у двох навантаженнях

	Метод	Метод	Запропонований метод
Зображення з текстом			
Зображення з текстом та картинками			

Результати, наведені в табл. 2, можуть дати чітке розуміння того, що середня точність генерації програмного коду у запропонованому алгоритмі є більшою на 5%, що дає змогу більш точніше генерувати програмний код, а також створювати його адаптивність та крос-браузерність.

4.3. Висновки до розділу 4

У даному розділі проведено аналіз розробленого методу генерації програмного коду за допомогою нейронних мереж на базі вхідного зображення. Метод дозволяє розв’язати поставлену задачу, даючи змогу створювати веб-сторінки, використовуючи тільки клієнтський інтерфейс.

Проведено аналіз якості генерації коду. Також проведено тестування роботи алгоритму в двох навантаженнях. Було виділено 4 основні показники: загальний час роботи алгоритму та точність генерації коду.

Завдяки використанню створеного програмного забезпечення було проведено аналіз показників якості генерації програмного коду запропонованим методом. Запропонований метод показав кращі на 10% результати зменшення часу роботи програмного забезпечення. Збільшено точність генерації програмного коду на 5%. Запропонований метод дозволяє генерувати адаптивний та крос-браузерний програмний код і має графічний інтерфейс користувача, чого немає у існуючих аналогів.

5. ПОБУДОВА БІЗНЕС-МОДЕЛІ

5.1. Опис проблеми та дерево проблем

5.1.1. *Анотація проекту*

Проект спрямований на розроблення та тестування методу відтворення коду веб-сторінки з вхідного дизайн-зображення за допомогою нейронної мережі та методів глибокого навчання. Розробка передбачає вирішення проблеми зменшення часу, необхідного для написання коду для повноцінного веб-сайту. Тестування передбачає створення різних видів симуляцій (мінімум два) роботи нейронної мережі для різних бібліотек та для різних пристроїв, що дасть змогу нейронній мережі працювати на будь-якому комп'ютері, адже нейронна мережа не знаходиться на присторії користувача, а тільки керується за допомогою користувацького інтерфейсу.

5.1.2. *Опис проблеми, на розв'язання якої спрямований проект*

Розробка способу відтворення HTML/CSS верстки веб-сайту з вхідного дизайн-макету за допомогою нейронної мережі полягає у знаходженні та розробленні програмного засобу, що за допомогою методу машинного навчання зможе в режимі реального часу проаналізувати вхідний макет-шаблон зображення, та за допомогою нейронної мережі відтворити безпосередньо HTML-код та каскадну таблицю стилів (CSS). Зрозуміло, що спочатку треба зібрати велику кількість вхідних дизайн-шаблонів зображень, з яких утворити дата-сет, який потрібен для навчання нейронної мережі.

В сучасному світі користувач може створити веб-сайт декількома способами. Перший спосіб полягає в безпосередньому написанні коду на будь-якій мові програмування, другий спосіб полягає в використанні так званих конструкторів веб-сайтів або інтернет-магазинів, а третій спосіб полягає в купівлі або замовленні веб-сайту у компанії. Також для підтримки

такого сайту потрібен фахівець, що має технічні знання, котрі необхідні для написання програмного коду. Головна проблема полягає в написанні простого веб-сайту за короткий період часу. Описаний метод відтворення сайтів гарантовано пришвидшить час створення невеликих веб-сайтів, адже користувачу треба лише намалювати дизайн-макет потрібної йому сторінки і нейронна мережа зробить все за нього.

Основна спрямованість проекту – створення алгоритму, який буде працювати швидше, ніж всі існуючі аналоги за рахунок розбиття каскадної таблиці стилів на декілька стильових файлів, які за допомогою автоматичного збірника на виході будуть збиратися в один файл, буде мати більшу точність відтворення, що використовуватиме унікальні дата-сети та методи машинного навчання, зменшить навантаження на сервери та на фізичні ресурси.

5.1.3. *Мета та завдання проекту відповідно до проблеми*

Метою проекту є створення веб-сторінки за допомогою нейронної мережі та методу глибокого машинного навчання на базі вхідного зображення-шаблону з урахуванням недоліків існуючих аналогів. Для цього необхідно виділити вже існуючі методи автоматичного створення веб-сторінок, виявити їх недоліки та переваги та обрати шлях розвитку проекту відповідно до виявлених переваг та недоліків. Рішення дозволить прискорити створення дата-сету, розподілить навантаження на кожний елемент нейронної мережі, що, в свою чергу, прискорить основний алгоритм. Також метою проєкту є створення симулятору для отримання статистичної інформації для порівняння з аналогами винайденого способу. Наступними кроком є публікація отриманих результатів та знаходження інвестицій для впровадження програмного продукту в масове використання. Залучення сторонніх розробників методом надання API у користування розробникам не для комерційних цілей дасть можливість використовувати та розвивати базу коду та функціональності реалізованого способу.

В основі методу є використання згорткової нейронної мережі та методу глибокого машинного навчання для навчання нейронної мережі розпізнавати різні елементи, зображені на вхідному файлі. Таким чином, спочатку треба “згодувати” нейронній мережі велику кількість вхідних даних та ряд правил, для відтворення зображення. Після цього ми отримаємо дата-сет прикладів та правил, на базі яких алгоритм і буде створювати веб-сайт.

5.2. Зацікавлені сторони

5.2.1. Зацікавлені сторони проекту

Зацікавлені сторони можна умовно поділити на такі групи: компанії, звичайні користувачі, розробники та міні сервіси (види підприємств або автоматизованих програмних комплексів, котрі створюють конструктори веб-сайтів або створюють однотипні веб-сайти у великій кількості).

1. Компанії. Тут зібрані особи, що зацікавлені у якісному продукті або рішенні та зацікавлені в розробленні добре діючого та продуктивного прототипу, способу або рішення, що може бути імплементоване для збільшення прибутку. На перших етапах компанії не зацікавлені в інвестуванні, але в подальшому можуть стати доволі потужним гравцем на ринку даного продукту. Це всі, хто зацікавлений саме у отриманні доступу до використання програмного методу, тобто «цільова аудиторія» методу. До неї входять усі фірми, підприємства, корпорації, заклади, що працюють над створенням веб-сайтів у тому чи іншому вигляді, а також можуть бути майданчиком з надання послуг користувачам або іншим компаніям.

2. Звичайні користувачі. Звичайні користувачі відіграють невелику роль в розвитку додатку, кожен користувач за допомогою програмного продукту та ноутбуку може створювати прості веб-сторінки з вхідного зображення. Але на початкових етапах запуску програмного продукту саме завдяки користувачам буде створена база даних та дата-сет для нейронної

мережі. Що прискорить швидкодію алгоритму перед запуском програмного продукту для інших зацікавлених сторін.

3. Розробники. Розробники відіграють велику роль в розвитку додатку. Саме розробники кожен день створюють велику-кількість веб-сайтів. Вони виконують однотипну роботу з дня в день. За допомогою програмного продукту, розробники зможуть набагато швидше робити свою роботу, автоматизувавши процес створення веб-сайту, що дасть їм змогу використати вільний час для стилізації автоматично створеного веб-сайту.

4. Міні-сервіси. До міні-сервісів входять маленькі IT або дизайн студії, які так само, як і розробники, створюють веб-сайти на замовлення. Також до цього пункту можна віднести всіх розробників та фрилансерів, які надають ті ж самі послуги від обличчя фірми.

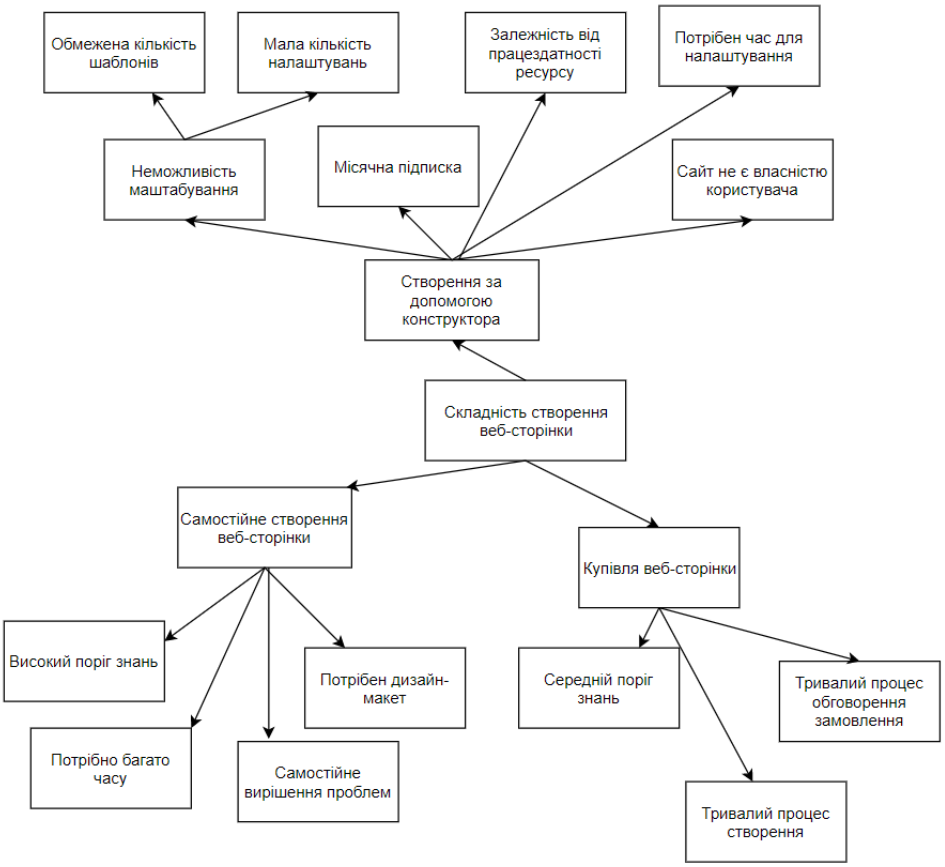


Рис. 24. Дерево проблем проєкту

У табл. 3 зведено усі групи зацікавлених сторін, їх інтереси, та вплив (міра зацікавленості у вирішенні наявних проблем).

Таблиця 3

Аналіз зацікавлених сторін проекту

Група зацікавлених осіб	Інтереси групи в проекті	Умови довгого співробітництва з проектом
Компанії	Забезпечення стабільної працездатності алгоритму	Модернізація, та стабільна робота алгоритму. Розвиток алгоритму.
Звичайні користувачі	Швидке створення веб-сторінки с вхідного шаблону-зображення	Нові функції. Розвиток алгоритму. Підтримка.
Розробники	Доступ до API. Зменшення витраченого часу на повсякденну розробку.	Модернізація, та стабільна робота алгоритму. Розвиток алгоритму.
Міні-сервіси	Стабільна робота алгоритму. Доступ до API.	Модернізація, та стабільна робота алгоритму. Розвиток алгоритму.

Таблиця 4

Зацікавлені сторони проекту. Загальна оцінка

Група зацікавлених осіб	Важливість	Зацікавленість	Оцінка	Сумарно
Компанії	8	8	Висока	64
Звичайні користувачі	4	4	Низька	16

Розробники	10	9	Дуже висока	90
Міні-сервіси	7	8	Висока	56

5.3. Комерційне рішення. Основні характеристики

Відповідно до вище зазначених проблем, можна описати кінцевий продукт, що має їх вирішувати. Даний програмний продукт буде реалізовувати описаний у попередніх розділах алгоритм створення веб-сторінок на базі шаблонного зображення за допомогою нейронної мережі. Даний алгоритм дозволить швидко створювати веб-сторінку без високого технічного рівня знань. Розробники зможуть автоматизувати процес роботи та витратити свій час на вдосконалення дизайну або ядра веб-сайту. По-перше, алгоритм не є ресурсоємним, що означає, що кожен користувач зможе користуватися ним без дорогого обладнання. По-друге, простота використання алгоритму зможе привабити нових користувачів за малий період часу. Саме тому слід підкреслити, що, в першу чергу, клієнтами продукту стануть компанії та міні-сервіси, а співпраця буде побудована на моделі співробітництва бізнесу та бізнесу, або як він ще називається B2B. В другу чергу, клієнтами стануть розробники веб-ресурсів та звичайні користувачі, а співпраця з ними відповідно буде побудована на моделі бізнесу для клієнта.

Даний програмний продукт повинен буте простим у використанні. Повинен відповідати новітнім вимогам розробки сучасних веб-ресурсів, та легко інтегруватись в нейронні мережі компаній користувачів. Також повинен бути наданий API розробникам, компаніям та міні-сервісам. Також разом з програмним продуктом йде технічна документація, що дасть змогу користувачам розібратися у всіх процесах роботи алгоритму, що зведе до мінімуму доцільність наймання спеціалістів для підтримки програмного продукту. Зважаючи на вище сказане, можна зробити висновок, що проблем

з інтеграцією з існуючими системами виникати не повинно, або вони повинні швидко і легко вирішитися за допомогою невеликої кількості спеціалістів.

5.4. Конкурентні переваги рішення

Нейронні мережі та методи глибокого машинного навчання є досить молодою сферою. Але все одно за останні роки було створено декілька реалізацій створення веб-сайтів за допомогою нейронних мереж. Такі реалізації були створенні в великих фірмах та не використовуються публічно. Також ці реалізації не є програмними продуктами, вони працюють за допомогою великих ресурсів, що мають фірми. Працювати з ними можуть тільки розробники. Для того, щоб запустити таку нейронну мережу на своєму комп'ютері, потрібні процесор та потужна відеокарта, а в деяких випадках і не одна. Рішення конкурентів мають багато недоліків, які були описані в пункті 5.1, майже всі недоліки вирішуються за допомогою запропонованого методу розподілення навантаження на збірник каскадної таблиці стилів. Разом з цим слід сказати, що даний алгоритм працює на згортковій нейронній мережі, котра з кожним запитом навчається, це означає, що з кожним новим запитом користувача, програмний продукт буде працювати все швидше і швидше. В теорії після декількох тисяч ітерацій, програмний продукт буде працювати швидше на 20-30 відсотків, ніж існуючі аналоги. Також після виходу програмного продукту на ринок, великі компанії та розробники отримають API для роботи безпосередньо з алгоритмом, що ще прискорить алгоритм, тим самим, розробники з усього світу матимуть можливість адаптувати алгоритм під свої задачі та створювати нову функціональність.

Отже, конкурентними перевагами програмного продукту, що пропонується, є:

- Зменшення навантаження на графічні та фізичні процесори.

- Збільшення швидкості відтворення веб-сайту.
- Збільшена можливість похибки в вхідному шаблоні.
- Час обчислення не залежить від характеристик обладнання.
- Доступність програмного продукту звичайним користувачам.
- Простота у використанні.
- Низький поріг знань для користування.

5.5. Клієнти. Сегменти ринку споживання

Як вже зазначалося вище, основними клієнтами даного програмного забезпечення є компанії та розробники.

На сьогоднішній день розроблення програмного забезпечення є ключовим фактором інформаційних технологій в цілому, що забезпечує працездатність і споживчу цінність ІТ. У цьому підпункті аналізується існуючий сьогодні світовий ринок розроблення програмного забезпечення, виділяються окремі сегменти ПЗ (платформенне, прикладне, мережеве, вбудоване ПЗ), підкреслюється низька капіталомісткість і високий ступінь диверсифікації ринку. Необхідно враховувати, що говорити про таку сегментацію можна тільки в рамках конкретного історичного зрізу, так як темпи розвитку інформаційних технологій призводять до суттєвих змін сегментації. В даному випадку можна відокремити три основні види сегментації: сегментація за критеріями досвіду праці з послугою і розмір бізнесу, сегментація за критеріям розміру бізнесу і сферою діяльності компанії клієнта та сегментацію за критерієм потреби до сервісу і важливістю послуги для бізнесу. Сегментація ринку за ознакою типу програмного продукту спричинена інженерним фактором. Це значить, що програмний продукт за допомогою нейронної мережі створює тільки клієнтську частину веб-сайту. Це значить, що програмний продукт створює тільки видиму частину сайту, а саме тільки HTML-код та CSS-код. На даний час є неможливим автоматичне створення ядра веб-сайту або фреймворку.

Це означає, що розробникам все одно треба буде самотужки писати ядро сайту або різну анімацію.

Також сегментацію ринку можна провести за видом веб-ресурсів, які зможе відтворити програмний продукт. Існують односторінкові, так звані “landing pages”, інтернет-магазини, односторінкові застосунки та багато іншого. Landing page – це будь-яка сторінка, яка закликає користувача щось зробити. Наприклад, підписатися на розсилку, купити квиток на конференцію, запросити кошторис або просто завантажити презентацію. На відміну від сайту, на лендінг користувачеві пропонують зробити щось одне. Наприклад, на сайт компанії приходять різні люди. Хтось хоче просто дізнатися про компанії, інші хочуть дізнатися інформацію про продукти. Але на сторінку з вакансією перейдуть ті, кому вона цікава. Якщо сторінка з вакансією буде створена за певними правилами і містити в собі пропозицію (наприклад, відгукнутися на вакансію), її теж можна вважати односторінковим лендінгом. Це так звані цільові сторінки. Перехід на цільові сторінки часто здійснюється з соціальних медіа, email-розсилок і рекламних кампаній в пошукових системах. Головним завданням таких сторінок є конвертація відвідувача в покупця або клієнта компанії, спонукання до цільового дії. Аналіз дій користувачів на цільовій сторінці дозволяє маркетологам визначити успішність реклами. Основною перевагою програмного продукту в даному сегменті є швидке створення таких односторінкових сайтів, що дає змогу користувачу одразу почати свій бізнес.

Також аналогом односторінкових сайтів є так звані односторінкові застосунки. Вони дуже різняться між собою. Сьогодні, популяризуючи сучасні фреймворки JavaScript, такі як React, програма зазвичай створюється як додаток для однієї сторінки: користувач завантажуєте код програми (HTML, CSS, JavaScript) лише один раз, і коли він взаємодіє з програмою, JavaScript перехоплює події веб-переглядача і замість того, щоб робити новий запит на сервер, який потім повертає новий документ, клієнт

запитує деякий файл у форматі JSON або виконує дію на сервері, але сторінка, яку бачить користувач, ніколи не буде повністю знищена, і виконує функції настільного додатку. Програми для однієї сторінки вбудовані в JavaScript (або, принаймні, зібрані в JavaScript) і працюють у браузері. Технологія завжди однакова, але філософія і деякі ключові компоненти того, як працює програма, різні. Ось приклад відомих компаній, у котрих сайти – це SPA:

- Google.
- Apple Maps.
- Facebook.
- Twitter.
- Google Drive.

Також слід додати веб-застосунки. Істотною перевагою побудови веб-застосунків для підтримки стандартних функцій браузера є те, що функції повинні виконуватися незалежно від операційної системи клієнта. Замість того, щоб писати різні версії для Microsoft Windows, Mac OS X, GNU/Linux й інших операційних систем, застосунок створюється один раз для довільно обраної платформи і на ній розгортається. Проте різна реалізація HTML, CSS, DOM й інших специфікацій в браузерах може викликати проблеми при розробці веб-застосунків і подальшої підтримки. Крім того, можливість користувача налаштовувати багато параметрів браузера (наприклад, розмір шрифту, кольори, відключення підтримки сценаріїв) може перешкоджати коректній роботі застосунку.

Отже, для співпраці найбільш перспективними є великі компанії, що мають спеціалізацію зі створення односторінкових веб-сайтів або односторінкових веб-застосунків, бо саме компанії є найбільш привабливими для бізнесу та приваблені нових клієнтів.

5.6. Унікальна ціннісна пропозиція

Цінність – це розуміння користі, яку покупець отримує від продукту. З такої позиції можна чітко охарактеризувати цінність продукту для кожної зацікавленої сторони або для кожного потенційного користувача, проаналізувати ці відомості та отримати детальну інформацію.

Так як цінність є суб’єктивною величиною, то далі неведена цінність відносно зацікавлених осіб.

- Компанії.
    - Забезпечити стабільну працездатність програмного продукту. Доступ до API.
    - Розпаралелювання виконуваних обмінів інформацією через з’єднання, а саме автоматичне розпаралелювання конвертування коду в багатопотоковий, що працює на паралельному комп'ютері, наприклад, на SMP або NUMA машині та перенесений на передатчик зв’язку. Повне розпаралелювання послідовних з’єднань залишається занадто складним завданням, що вимагає складніших видів аналізу програм.
- Звичайні користувачі
    - Багато функцій в програмному продукті. Дуже простий інтерфейс. Швидкодія та стабільність в роботі.
- Розробники
    - Стабільність.
    - Модернізація.
    - Кросплатформеність.
    - Технічна підтримка.
    - Технічна документація.
    - Підтримка long-pooling з'єднань.
- Міні-сервіси

- Масштабування для використання командами.

Отже, основною унікальною ціннісною пропозицією є розроблений програмний продукт, що враховує всі потреби зацікавлених сторін.

5.7. Доходи та витрати

Дана розробка має декілька потенційних джерел доходів, які наведені в табл. 3.

Мікротранзакції – це популярна бізнес-модель розповсюдження завантажуваного контенту або доступу до надаваних послуг за невеликими цінами (зазвичай в межах 10$). Така модель особливо поширена в різних MMORPG-іграх і часто замінює собою плату у вигляді підписки. Також подібна модель оплати застосовується при поширенні контент-послуг в мережах стільникового зв'язку (мелодії, картинки, програми для мобільних телефонів і комунікаторів). Невисокий розмір оплати знижує психологічний бар'єр перед покупкою у потенційного покупця.

1. Контекстний аналіз на основі трафіку виявлення і пошук, визначення властивостей та характеристик на основі зібраних статистичних даних або емпіричних досліджень окремих об'єктів або явищ. Використовуються з метою встановлення логічних закономірностей, які впливають на досліджувані об'єкти або явища, і пошуку переваг та вразливостей, які можуть виявлятися під впливом факторів.
2. Контекстний аналіз на основі місцеположення користувача.

Імплементування рішень в сферу логістики щодо покращення якості підтримки постійних клієнтів. Монетизація на мікротранзакціях та продажу ліцензій. Дана ліцензія передбачає використання закритого, власницького або пропрієтарного ПЗ.

Детальніше ознайомитися з витратами на реалізацію проєкту та прогнозованим прибутками можна з табл. 5.

Таблиця 5

Таблиця доходів та витрат продукту за перші 6 місяців

Найменування витрат	1-й місяць, т.$	2-й, т.$	3-й, т.$	4-й, т.$	5-й, т.$	6-й, т.$
Загальні витрати	3	3	3	3	3	3
ЗП		15	15	15	15	15
Витрати	3	18	18	18	18	18
Заплановані прибутки				60	60	60
Результат (без оподаткування)	–3	–18	–18	42	42	42

Таблиця 6

Таблиця доходів та витрат продукту за другі 6 місяців

Найменування витрат	7	8	9	10	11	12	Результати
Загальні витрати	3	3	3	3	3	3	36
ЗП	20	20	20	20	20	20	195
Витрати	23	23	23	23	23	23	231
Заплановані прибутки	60	60	60	60	60	60	540
Результат (без оподаткування)	42	42	42	42	42	42	339

5.8. Бізнес модель

Бізнес-модель – це те, як ми організовуємо нашу діяльність, щоб надати замовнику обіцяну цінність. Це те, як ми організовуємо наші

внутрішні операції, щоб виробляти і доставляти цінність замовнику. Нашими клієнтами можуть бути або кінцеві споживачі (якщо у ланцюжку створення цінності ми знаходимося ближче до кінцевої ланки, клієнта), або інші організації або суб'єкти ланцюжка залежно від того, де ми знаходимося у ньому.

Узагальнимо, все написане вище у звичайну бізнес-модель у вигляді canvas.

Споживачі: великі компанії, розробники, міні-сервіси та звичайні користувачі.

Проблема: Немає простих, не ресурсоємних рішень; високий вхідний поріг знань для використання рішення; велика затратність; відсутність технічної документації; мала швидкодія.

Рішення: програмне забезпечення, що створює веб-сайт на базі вхідного шаблону-зображення за допомогою нейронної мережі та методу глибокого машинного навчання.

Унікальна ціннісна пропозиція: розроблений програмний продукт, що враховує всі потреби зацікавлених сторін. Зменшення часу, потрібного на рутинне створення шаблонних сайтів.

Потоки доходів: доходи від продажу ліцензій та доходи з мікротранзакцій.

Структура витрат: початковий персонал для розроблення програмного продукту, утримання персоналу для надання технічної підтримки, заробітня платня та соціальні виплати. Оплата за оренду офісу. Оплата комунальних послуг.

Бізнес-модель наведена у зведеному вигляді у табл. 7.

Отже, зважаючи на дані у табл. 5, можна зробити висновок, що запропонований проєкт, який реалізує описаний у дисертацій алгоритм створення веб-сайтів, має перспективи у своїй подальшій реалізації. Звичайно, проведений аналіз не враховує всіх ризиків та факторів, таких як специфіка оподаткування у країні ведення бізнесу, проте навіть наявних

досліджень достатньо, щоб прогнозувати комерційний успіх продукту та його окупаємість.

Таблиця 7

Канва бізнес-моделі

<table>
<tr>
<td>Проблема
Немає простих, не ресурсоємних рішень;
високий вхідний поріг знань для використання рішення;
велика затратність;
відсутність технічної документації;
мала швидкодія.</td>
<td>Рішення
Програмне забезпечення, що створює веб-сайт на базі вхідного шаблону-зображення за допомогою нейронної мережі та методу глибокого машинного навчання.</td>
<td>Унікальна ціннісна пропозиція
Розроблений програмний продукт, що враховує всі потреби зацікавлених сторін. Зменшення часу, потрібного на рутинне створення шаблонних сайтів.</td>
<td>Ключові метрики
Кількість проданих ліцензій та кількість зроблених мікротранзакцій.</td>
</tr>
<tr>
<td colspan="3">Структура витрат</td>
<td>Потоки доходів</td>
</tr>
</table>

початковий персонал для розроблення програмного продукту, утримання персоналу для надання технічної підтримки, заробітня платня та соціальні виплати. Оплата за оренду офісу. Оплата комунальних послуг.	доходи від продажу ліцензій та доходи з мікротранзакцій.

5.9. Висновки до розділу 5

На основі проведеного дослідження побудовано працюючу бізнес-схему з прибутками, виявлено наявні проблеми та підсумовано їх у відповідному дереві проблем.

Також разом із проблемами, виділено основні зацікавлені сторони, методи вирішення існуючих проблем та ступінь впливу зацікавлених сторін на вирішення проблем.

Розроблений програмний метод має свої джерела прибутку та канали збуту продуктів, в основу методу покладено роботу та подальше навчання нейронної мережі, що призведе до збільшення швидкодії програмного продукту.

Проведено аналіз майбутніх клієнтів, досліджено сегменти ринку споживання. Це дозволило спрогнозувати потенційні доходи та витрати на реалізацію продукту. У результаті проведеного аналізу описана бізнес-модель, що обґрунтовує доцільність реалізації даного продукту та прогнозує його потенційну окупаємість та прибутковість в подальшому.

Метод легко масштабується як у вигляді бібліотеки, так і як веб-застосунок для створення веб-сайтів на базі вхідного шаблону-зображення для забезпечення вимог клієнтів. На даний момент складається патентна документація, що пришвидшить імплементування даної розробки в продуктивне середовище.

ВИСНОВКИ

У даній магістерській дисертації було поставлено за мету дослідити існуючі методи генерації програмного коду, а саме методи створення програмного коду за допомогою методів машинного навчання, що надають можливість згенерувати програмний код веб-сторінки, базуючись лише на вхідному зображенні. Результатом досліджень є розроблене програмне забезпечення на основі запропонованого методу генерації програмного коду з використанням нейронної мережі на базі вхідного зображення, яким можна керувати за допомогою користувацького інтерфейсу.

У першому розділі проаналізовано існуючі рішення для генерації програмного коду за допомогою методів машинного навчання. Виявлено їх переваги та недоліки. Для проведення аналізу було створено порівняльну таблицю. На основі досліджень та результатів аналізу було сформовано вимоги до програмного методу. Основними вимогами стали: створення користувацького інтерфейсу для керування програмною реалізацією методу; можливість налаштовувати роботу програмного забезпечення в користувацькому інтерфейсі; збільшення швидкодії програмного методу шляхом одночасної роботи генеративно-змагальної та рекурентної нейронних мереж; генерація кросбраузерного та адаптивного програмного коду.

У другому розділі даної магістерської дисертації детально описано запропонований програмний метод. З огляду на недоліки існуючих аналогів було вирішено не модифікувати їх, а створити власний програмний метод. Особливістю запропонованого методу є одночасна робота генеративно-змагальної та рекурентної нейронної мережі. Для досягнення цього процесу використаний власний метод створення DSL-дерева.

У третьому розділі описано стек технологій та розроблене програмне забезпечення, яке складається з двох основних частин, а саме з front-end та back-end частин. Завдяки тому, що користувацький інтерфейс був розроблений за допомогою фреймворку ReactJS, користувач має змогу налаштовувати складний процес роботи нейронних мереж без можливості зламати щось. Робота користувацького інтерфейсу при переході по роутам проекту відбувається без перезавантаження сторінки. Серверна частина розроблена з використанням мови програмування Python та бібліотеки Keras. Також впроваджений власний метод генерації DSL-дерев, що дало змогу змусити GAN та RNN працювати одночасно, що в свою чергу, дало можливість реалізувати всі поставлені задачі другого розділу.

У четвертому розділі даної дисертації описано методологію тестування для розроблюваної системи, проаналізовано результати роботи та визначено напрями для подальшого вдосконалення системи у майбутньому. Базуючись на результатах тестування, можна заявити, що розроблений програмний метод працює на 10% швидше, ніж існуючі аналоги. Також слід додати, що був реалізований ряд функціональних можливостей, котрих немає у існуючих аналогів і це не вплинуло на загальний час роботи.

За результатами проведеної роботи в останньому розділі побудовано бізнес-модель кінцевого продукту, що описує ключові моменти в організації діяльності, пов’язаної з поширенням розробленого програмного методу генерації програмного коду веб-сторінок за допомогою нейронної мережі на базі вхідного зображення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Tony Yiu, Understanding Neural Networks [Електронний ресурс] / Towards Data Science — Режим доступу: https://towardsdatascience.com/understanding-neural-networks-19020b758230. — Дата доступу: 3.12.2019.
2. D. Bahdanau, K. Cho, and Y. Bengio, Neural machine translation by jointly learning to align and translate [Електронний ресурс] / Arxiv Org. — Режим доступу: https://arxiv.org/pdf/1409.0473.pdf. — Дата доступу: 3.12.2019.
3. Michael Nielsen, Neural Networks and Deep Learning [Електронний ресурс] / Neural Networks and Deep Learning — Режим доступу: http://neuralnetworksanddeeplearning.com/index.html. — Дата доступу: 3.12.2019.
4. S. Hochreiter and J. Schmidhuber, Long short-term memory. Neural computation [Електронний ресурс] / Bioinf At. — Режим доступу: https://www.bioinf.jku.at/publications/older/2604.pdf. — Дата доступу: 3.12.2019.
5. T. Tieleman and G. Hinton, Divide the gradient by a running average of its recent magnitude. Neural computation [Електронний ресурс] / Coursera. — Режим доступу:

https://www.cs.toronto.edu/lecture_slides_lec6.pdf. — Дата доступу: 3.12.2019.

6. A. L. Gaunt, M. Brockschmidt, A probabilistic programming language for program induction [Електронний ресурс] / Arxiv Org. — Режим доступу: https://arxiv.org/pdf/1612.00817.pdf. — Дата доступу: 3.12.2019.
7. I. Goodfellow, J. Pouget-Abadie, Generative adversarial nets [Електронний ресурс] / Arxiv Org. — Режим доступу: https://arxiv.org/pdf/1406.2661.pdf. — Дата доступу: 3.12.2019.
8. K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition [Електронний ресурс] / Arxiv Org — Режим доступу: https://arxiv.org/pdf/1409.1556.pdf. — Дата доступу: 3.12.2019.
9. A. Karpathy and L. Fei-Fei, Deep visual-semantic alignments for generating image descriptions [Електронний ресурс] / Stanford Edu. — Режим доступу: https://cs.stanford.edu/people/karpathy/cvpr2015.pdf. — Дата доступу: 3.12.2019.
10. K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition [Електронний ресурс] / Arxiv Org. — Режим доступу: https://arxiv.org/pdf/1409.1556.pdf. — Дата доступу: 3.12.2019.
11. W. Zaremba, I. Sutskever, Recurrent neural network regularization [Електронний ресурс] / Arxiv Org. — Режим доступу: https://arxiv.org/pdf/1409.2329.pdf. — Дата доступу: 3.12.2019.
12. H. Zhang, T. Xu, Text to photo-realistic image synthesis with stacked generative adversarial networks [Електронний ресурс] / Arxiv Org. — Режим доступу: https://arxiv.org/abs/1612.03242. — Дата доступу: 3.12.2019.

13. A. L. Gaunt, A probabilistic programming language for program induction [Електронний ресурс] / Arxiv Org. — Режим доступу: https://arxiv.org/abs/1608.04428. — Дата доступу: 3.12.2019.
14. Eisenman, B. Learning React [Text] / Bonnie Eisenman. — 2nd edition. — Sebastopol: O'Reilly Media, 2017. — 242 p.
15. Kirupa, Understanding WebViews [Електронний ресурс] / Kirupa. — Режим доступу: https://www.kirupa.com/apps/webview.htm. — Дата доступу: 3.12.2019.
16. WebKit Org., The WebKit Opensource Project [Електронний ресурс] / WebKit Org. — Режим доступу: https://webkit.org/project/. — Дата доступу: 3.12.2019.
17. Garbade, M. Native vs. cross-platform app development: pros and cons [Електронний ресурс] / Dr. Michael J. Garbade. — Режим доступу: https://codeburst.io/native-vs-cross-platform-app-development-pros-and-cons-49f397bb38ac. — Дата доступу: 3.11.2019.
18. LeRoux, B. PhoneGap Beliefs, Goals, and Philosophy [Електронний ресурс] / Brian LeRoux. — Режим доступу: https://phonegap.com/blog/2012/05/09/phonegap-beliefs-goals-and-philosophy/. — Дата доступу: 2.11.2019.
19. Trice, A. PhoneGap Explained Visually [Електронний ресурс] / Andrew Trice. — Режим доступу: https://phonegap.com/blog/2012/05/02/phonegap-explained-visually/. — Дата доступу: 2.11.2019.
20. Wargo, J. Apache Cordova 4 Programming (Mobile Programming) [Text] / John M. Wargo. — 1st edition. — Boston: Addison-Wesley Profeccional, 2015. — 560 p.
21. Griffith, C. Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova [Text] / Chris Griffith. — 1st edition. — Sebastopol: O'Reilly Media, 2017. — 292 p.

22. Ionic, Core Concepts [Електронний ресурс] / Ionic. — Режим доступу: https://ionicframework.com/docs/intro/concepts. — Дата доступу: 2.11.2019.
23. Flutter, Flutter for iOS developers [Електронний ресурс] / Flutter. — Режим доступу: https://flutter.dev/docs/get-started/flutter-for/ios-devs. — Дата доступу: 2.11.2019.
24. Napoli, M. Beginning Flutter: A Hands On Guide to App Development [Text] / Varco L. Napoli. — 1st edition. — Birmingham: Wrox, 2019. — 528 p.
25. Aggarwal, R. 10 top Programming Languages in 2019 for Businesses [Електронний ресурс] / Ruchika Singh Aggarwal. — Режим доступу: https://codeburst.io/10-top-programming-languages-in-2019-for-developers-a2921798d652. — Дата доступу: 3.11.2019.
26. Google Inc., Building web apps [Електронний ресурс] / Google Inc. — Режим доступу: https://developer.android.com/guide/webapps/webview. — Дата доступу: 3.11.2019.
27. Kumar, S. How React Works? [Електронний ресурс] / Saket Kumar. — Режим доступу: https://www.codementor.io/saketkumar95/how-react-native-works-mhjo4k6f3. — Дата доступу: 3.11.2019.
28. Buckler, C. JavaScript HTML5 Programming [Text] / Craig Buckler. — 2nd edition. — Sebastopol: O'Reilly Media, 2012. — 241 p.
29. Morley, M. JSON-RPC 2.0 Specification [Електронний ресурс] / Matt Morley. — Режим доступу: https://www.jsonrpc.org/specification. — Дата доступу: 17.11.2019.
30. Turskyi, V. MOLE-RPC [Електронний ресурс] / Viktor Turskyi. — Режим доступу: https://www.npmjs.com/package/mole-rpc. — Дата доступу: 20.11.2019.

31. Facebook, Communication between native and React Native [Електронний ресурс] / Facebook. — Режим доступу: https://facebook.github.io/react-native/docs/communication-ios. — Дата доступу: 22.11.2019.
32. Vepsäläinen, J. SuviveJS – Webpack and React: From apprentice to master [Text] / Juho Vepsäläinen. — 2st edition. — Scotts Valley: CreateSpace Independent Publishing Platform, 2016. — 284 p.
33. Turskyi, V. Yet another JSON RPC Library? [Електронний ресурс] / Viktor Turskyi. — Режим доступу: https://docs.google.com/presentation/d/1NCmVJalBJp0Gliyc8KCdExHTl-xztDz3v50V4Y2k0bQ/edit#slide=id.g43ac3735fc_0_757. — Дата доступу: 22.11.2019.
34. Facebook, Out-of-Tree Platforms [Електронний ресурс] / Facebook. — Режим доступу: https://facebook.github.io/react-native/docs/out-of-tree-platforms. — Дата доступу: 30.11.2019.
35. Souders, S. Even Faster Web Sites: Performance Best Practices for Web Developers [Text] / Steve Souders. — 1st edition. — Sebastopol: O'Reilly Media, 2009. — 256 p.
36. Google Inc., About Android App Bundles [Електронний ресурс] / Google Inc. — Режим доступу: https://developer.android.com/guide/app-bundle. — Дата доступу: 30.11.2019.
37. ViroMedia, Overview [Електронний ресурс] / ViroMedia. — Режим доступу: https://docs.viromedia.com/docs/viro-platform-overview. — Дата доступу: 30.11.2019.
38. Gasston, P. The Modern Web: Multi-Device Web Development with HTML5, CSS3, and JavaScript [Text] / Peter Gasston. — 1nd edition. — San Francisco: No Starch Press, 2013. — 264 p.

39. Simpson, K. You Don’t Know JS: Async & Performance [Text] / Kyle Simpson. — 1nd edition. — Sebastopol: O’Reilly Media, 2015. — 296 p.

ДОДАТКИ

Додаток 1
Лістинг програми

Лістинг 1. generation.py

```
from __future__ import print_function
from __future__ import absolute_import

import os
import sys

from classes.Sampler import *
from classes.model.pix2code import *

argv = sys.argv[1:]

if len(argv) < 4:
    print("Error: not enough argument supplied:")
    print("generate.py <trained weights path> <trained model name> <input
image> <output path> <search method (default: greedy)>")
    exit(0)
else:
    trained_weights_path = argv[0]
    trained_model_name = argv[1]
    input_path = argv[2]
    output_path = argv[3]
    search_method = "greedy" if len(argv) < 5 else argv[4]

meta_dataset =
np.load("{}/meta_dataset.npy".format(trained_weights_path))
input_shape = meta_dataset[0]
output_size = meta_dataset[1]

model = pix2code(input_shape, output_size, trained_weights_path)
model.load(trained_model_name)

sampler = Sampler(trained_weights_path, input_shape, output_size,
CONTEXT_LENGTH)

for f in os.listdir(input_path):
    if f.find(".png") != -1:
        evaluation_img =
Utils.get_preprocessed_img("{}/{}".format(input_path, f), IMAGE_SIZE)

        file_name = f[:f.find(".png")]

        if search_method == "greedy":
            result, _ = sampler.predict_greedy(model,
np.array([evaluation_img]))
            print("Result greedy: {}".format(result))
        else:
            beam_width = int(search_method)
            print("Search with beam width: {}".format(beam_width))
            result, _ = sampler.predict_beam_search(model,
np.array([evaluation_img]), beam_width=beam_width)
            print("Result beam: {}".format(result))

        with open("{}/{}.gui".format(output_path, file_name), 'w') as
out_f:
            out_f.write(result.replace(START_TOKEN,
"").replace(END_TOKEN, ""))
```

Лістинг 2. imgToList.py

```
import os
import sys
import shutil

from classes.Utils import *
from classes.model.Config import *

argv = sys.argv[1:]

if len(argv) < 2:
    print("Error: not enough argument supplied:")
    print("convert_imgs_to_arrays.py <input path> <output path>")
    exit(0)
else:
    input_path = argv[0]
    output_path = argv[1]

if not os.path.exists(output_path):
    os.makedirs(output_path)

print("Converting images to numpy arrays...")

for f in os.listdir(input_path):
    if f.find(".png") != -1:
        img = Utils.get_preprocessed_img("{}/{}".format(input_path, f),
IMAGE_SIZE)
        file_name = f[:f.find(".png")]

        np.savez_compressed("{}/{}".format(output_path, file_name),
features=img)
        retrieve = np.load("{}/{}.npz".format(output_path,
file_name))["features"]

        assert np.array_equal(img, retrieve)

        shutil.copyfile("{}/{}.gui".format(input_path, file_name),
"{}/{}.gui".format(output_path, file_name))


print("Numpy arrays saved in {}".format(output_path))
```

Лістинг 3. createDataSet.py

```
import os
import sys
import hashlib
import shutil

from classes.Sampler import *

argv = sys.argv[1:]

if len(argv) < 1:
    print("Error: not enough argument supplied:")
    print("build_datasets.py <input path> <distribution (default: 6)>")
    exit(0)
else:
    input_path = argv[0]
```

```
distribution = 6 if len(argv) < 2 else argv[1]

TRAINING_SET_NAME = "training_set"
EVALUATION_SET_NAME = "eval_set"

paths = []
for f in os.listdir(input_path):
    if f.find(".gui") != -1:
        path_gui = "{}/{}".format(input_path, f)
        file_name = f[:f.find(".gui")]

        if os.path.isfile("{}/{}.png".format(input_path, file_name)):
            path_img = "{}/{}.png".format(input_path, file_name)
            paths.append(file_name)

evaluation_samples_number = len(paths) / (distribution + 1)
training_samples_number = evaluation_samples_number * distribution

assert training_samples_number + evaluation_samples_number == len(paths)

print("Splitting datasets, training samples: {}, evaluation samples:
{}".format(training_samples_number, evaluation_samples_number))

np.random.shuffle(paths)

eval_set = []
train_set = []

hashes = []
for path in paths:
    if sys.version_info >= (3,):
        f = open("{}/{}.gui".format(input_path, path), 'r',
encoding='utf-8')
    else:
        f = open("{}/{}.gui".format(input_path, path), 'r')

    with f:
        chars = ""
        for line in f:
            chars += line
        content_hash = chars.replace(" ", "").replace("\n", "")
        content_hash = hashlib.sha256(content_hash.encode('utf-
8')).hexdigest()

        if len(eval_set) == evaluation_samples_number:
            train_set.append(path)
        else:
            is_unique = True
            for h in hashes:
                if h is content_hash:
                    is_unique = False
                    break

            if is_unique:
                eval_set.append(path)
            else:
                train_set.append(path)

        hashes.append(content_hash)

assert len(eval_set) == evaluation_samples_number
assert len(train_set) == training_samples_number
```

```
if not os.path.exists("{}/{}".format(os.path.dirname(input_path),
EVALUATION_SET_NAME)):
    os.makedirs("{}/{}".format(os.path.dirname(input_path),
EVALUATION_SET_NAME))

if not os.path.exists("{}/{}".format(os.path.dirname(input_path),
TRAINING_SET_NAME)):
    os.makedirs("{}/{}".format(os.path.dirname(input_path),
TRAINING_SET_NAME))

for path in eval_set:
    shutil.copyfile("{}/{}.png".format(input_path, path),
"{}/{}/{}.png".format(os.path.dirname(input_path), EVALUATION_SET_NAME,
path))
    shutil.copyfile("{}/{}.gui".format(input_path, path),
"{}/{}/{}.gui".format(os.path.dirname(input_path), EVALUATION_SET_NAME,
path))

for path in train_set:
    shutil.copyfile("{}/{}.png".format(input_path, path),
"{}/{}/{}.png".format(os.path.dirname(input_path), TRAINING_SET_NAME,
path))
    shutil.copyfile("{}/{}.gui".format(input_path, path),
"{}/{}/{}.gui".format(os.path.dirname(input_path), TRAINING_SET_NAME,
path))

print("Training dataset:
{}/training_set".format(os.path.dirname(input_path), path))
print("Evaluation dataset:
{}/eval_set".format(os.path.dirname(input_path), path))
```

Лістинг 4. Training.py

```
import tensorflow as tf
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))

import sys

from classes.dataset.Generator import *
from classes.model.pix2code import *


def run(input_path, output_path, is_memory_intensive=False,
pretrained_model=None):
    np.random.seed(1234)

    dataset = Dataset()
    dataset.load(input_path, generate_binary_sequences=True)
    dataset.save_metadata(output_path)
    dataset.voc.save(output_path)

    if not is_memory_intensive:
        dataset.convert_arrays()

        input_shape = dataset.input_shape
        output_size = dataset.output_size

        print(len(dataset.input_images), len(dataset.partial_sequences),
len(dataset.next_words))
        print(dataset.input_images.shape,
dataset.partial_sequences.shape, dataset.next_words.shape)
```

```
    else:
        gui_paths, img_paths = Dataset.load_paths_only(input_path)

        input_shape = dataset.input_shape
        output_size = dataset.output_size
        steps_per_epoch = dataset.size / BATCH_SIZE

        voc = Vocabulary()
        voc.retrieve(output_path)

        generator = Generator.data_generator(voc, gui_paths, img_paths,
batch_size=BATCH_SIZE, generate_binary_sequences=True)

    model = pix2code(input_shape, output_size, output_path)

    if pretrained_model is not None:
        model.model.load_weights(pretrained_model)

    if not is_memory_intensive:
        model.fit(dataset.input_images, dataset.partial_sequences,
dataset.next_words)
    else:
        model.fit_generator(generator, steps_per_epoch=steps_per_epoch)

if __name__ == "__main__":
    argv = sys.argv[1:]

    if len(argv) < 2:
        print("Error: not enough argument supplied:")
        print("train.py <input path> <output path> <is memory intensive
(default: 0)> <pretrained weights (optional)>")
        exit(0)
    else:
        input_path = argv[0]
        output_path = argv[1]
        use_generator = False if len(argv) < 3 else True if int(argv[2])
== 1 else False
        pretrained_weigths = None if len(argv) < 4 else argv[3]

    run(input_path, output_path, is_memory_intensive=use_generator,
pretrained_model=pretrained_weigths)
```

Додаток 2
Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

ПРОГРАМНИЙ МЕТОД СТВОРЕННЯ ВЕБ-СТОРІНКИ ЗА ДОПОМОГОЮ НЕЙРОННОЇ МЕРЕЖІ НА БАЗІ ВХІДНОГО ЗОБРАЖЕННЯ

Виконав: Богуцький Д.Б., КП-81 мп
Науковий керівник: к.т.н., доцент, Олещенко Л. М.

Київ – 2019

Актуальність

Вже сьогодні технології глибинного навчання аналізують та створюють зовнішні інтерфейси веб-сайтів.

У майбутньому це прискорить розробку та знизить бар’єр для створення веб-застосунків різних типів складності.

Наукове завдання

Створення алгоритму машинного навчання для нейронної мережі, яка буде генерувати програмний код веб-сторінки на основі вхідного зображення.

Мета дослідження

Створити алгоритм відтворення веб-сторінки за допомогою нейронної мережі на базі вхідного зображення-шаблону з урахуванням недоліків існуючих аналогів.

Об’єкт дослідження

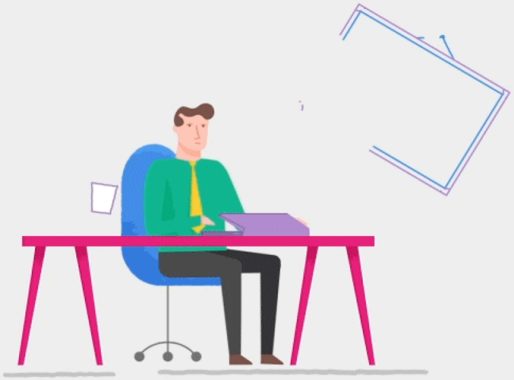
Процес веб-розробки з використанням алгоритму генерації програмного коду веб-сторінки на базі вхідного зображення.

Предмет дослідження

Сучасні алгоритми нейронних мереж машинного навчання для штучної автоматизації зовнішнього інтерфейсу.

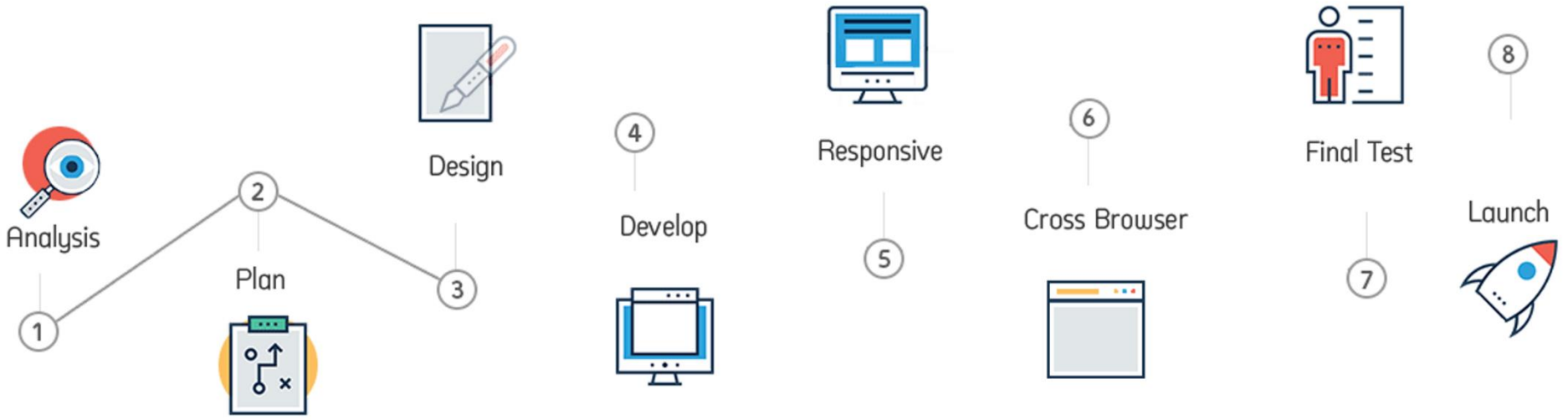
Проблематика

- Створення веб-сайтів зводиться до повсякденної рутинної роботи.
- Веб-розробники не використовують існуючі рішення генерації програмного коду адже їх:
  - складно налаштовувати;
  - складно масштабувати;
  - код не чутливий;
  - навантаження на систему дуже високе;
  - не доцільно використовувати у великих проектах;
  - технологія ще є “сирою”.



5

Процес створення веб-сайту



Задачі

#1

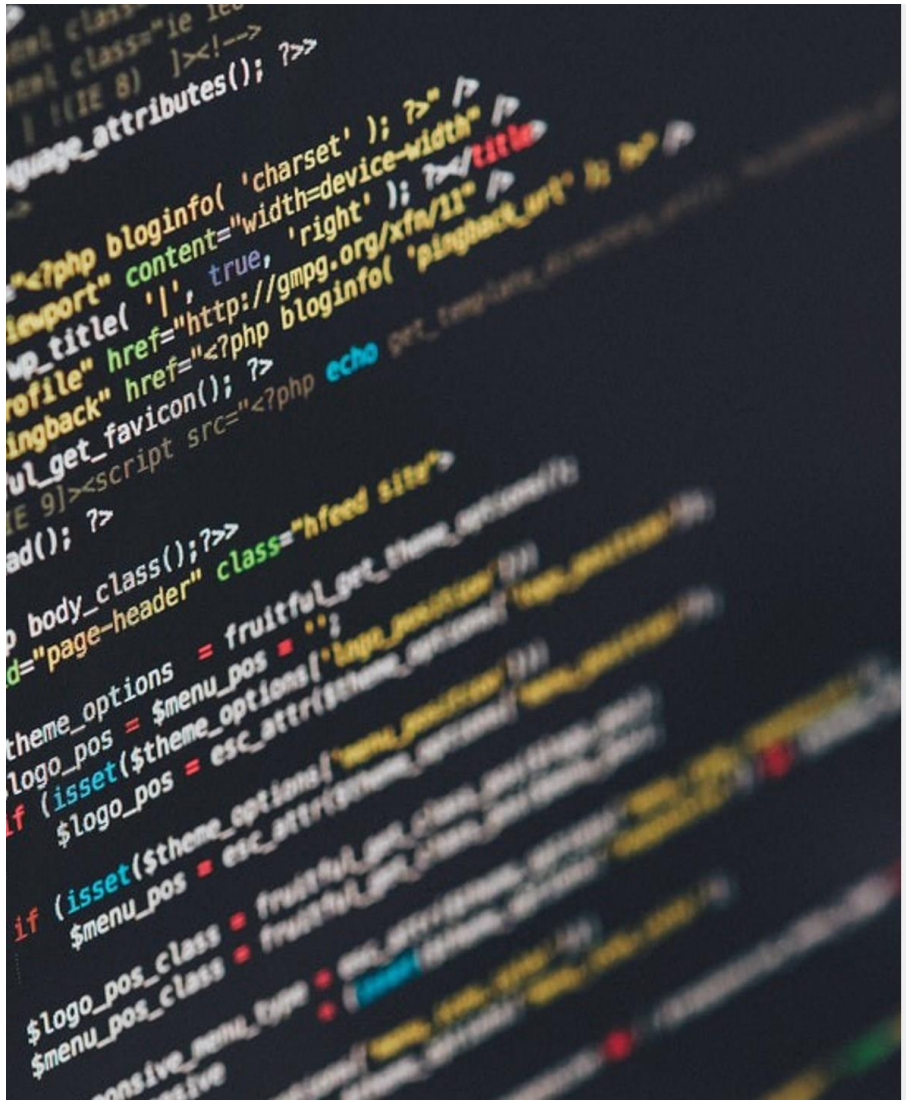
Аналіз варіантів архітектури та способів машинного навчання. Аналіз методів генерації програмного коду. Аналіз методів розпізнавання зображень.

#2

Розробка власного програмного методу генерації програмного коду веб-сторінки за допомогою нейронних мереж з вхідного зображення. Налаштування віддаленого серверу для обробки інформації. Створення користувацького інтерфейсу.

#3

Тестування та порівняння результатів дослідження. Створення стартапу на основі імплементації розробленого програмного методу.



Існуючі методи генерації коду

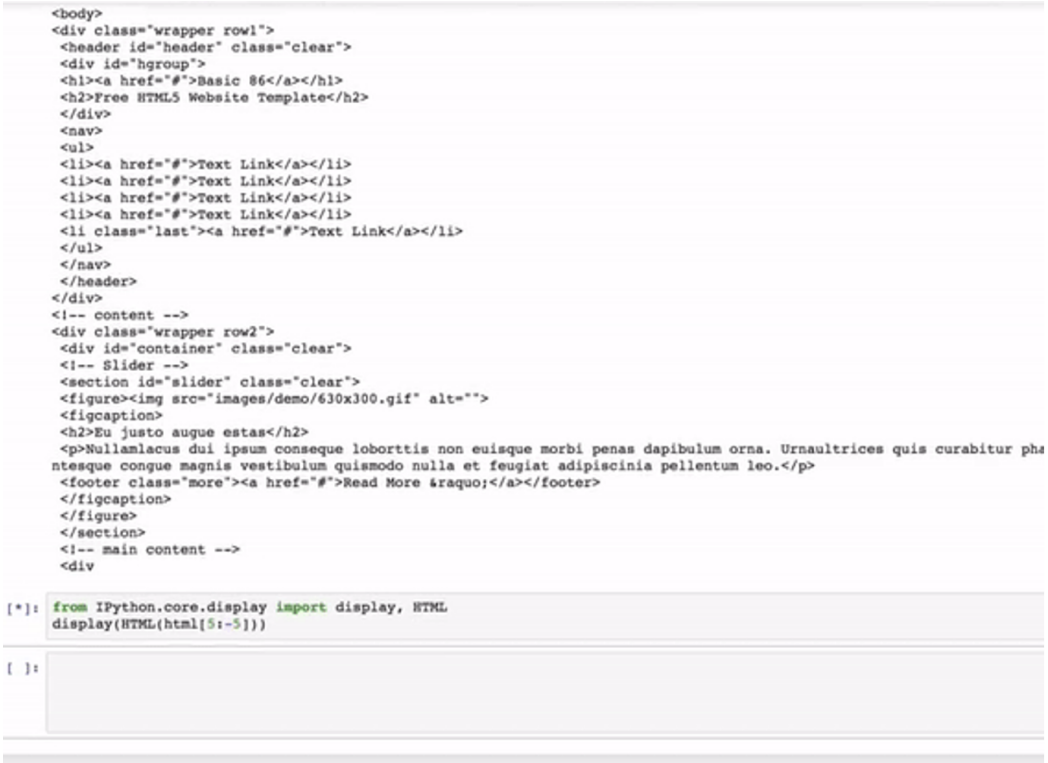
.DeepCoder

DEEPCODE

.pix2Code

uizard

8



Опис методу

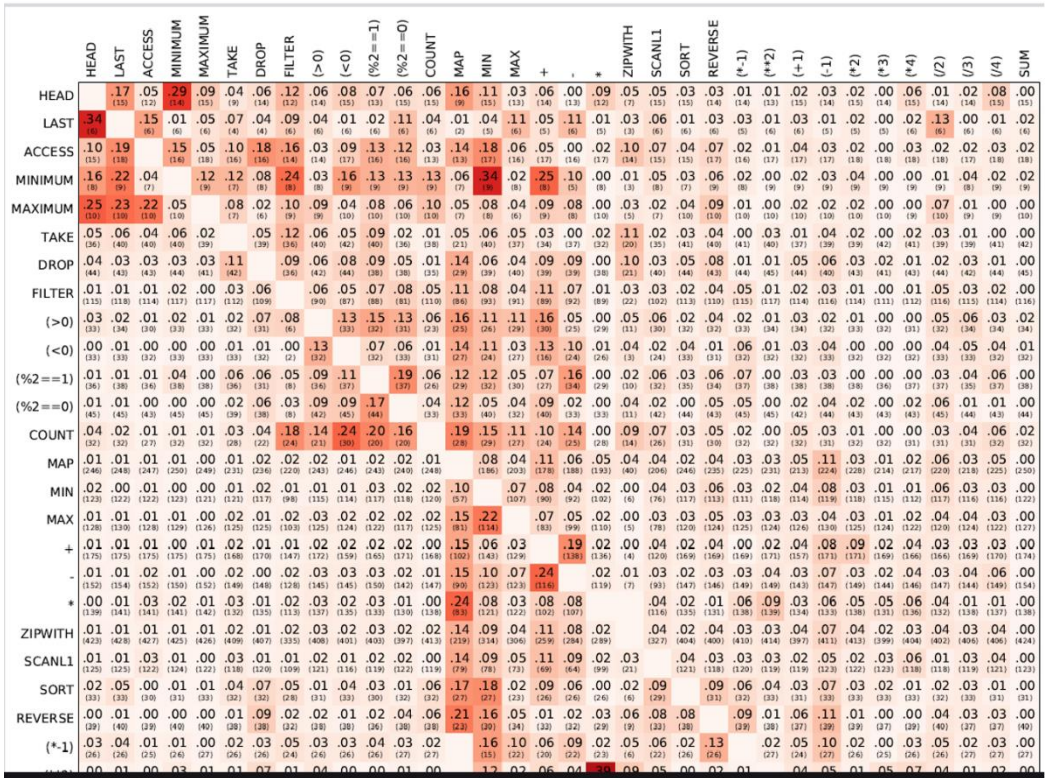
Нейронна мережа розпізнає вхідне зображення та генерує HTML & CSS код вихідного веб-сайту

Швидкість x4

Pix2Code

Недоліки

- Знання мови Python
- Високий вхідний бар’єр
- Обчислювальна потужність
- Відсутність користувацького інтерфейсу
- Не “просте” вхідне зображення
- Вихідний код:
  - не адаптивний;
  - не чутливий.



Опис методу

Метод запозичує рядки коду з других програм і створює свою власну унікальну програму

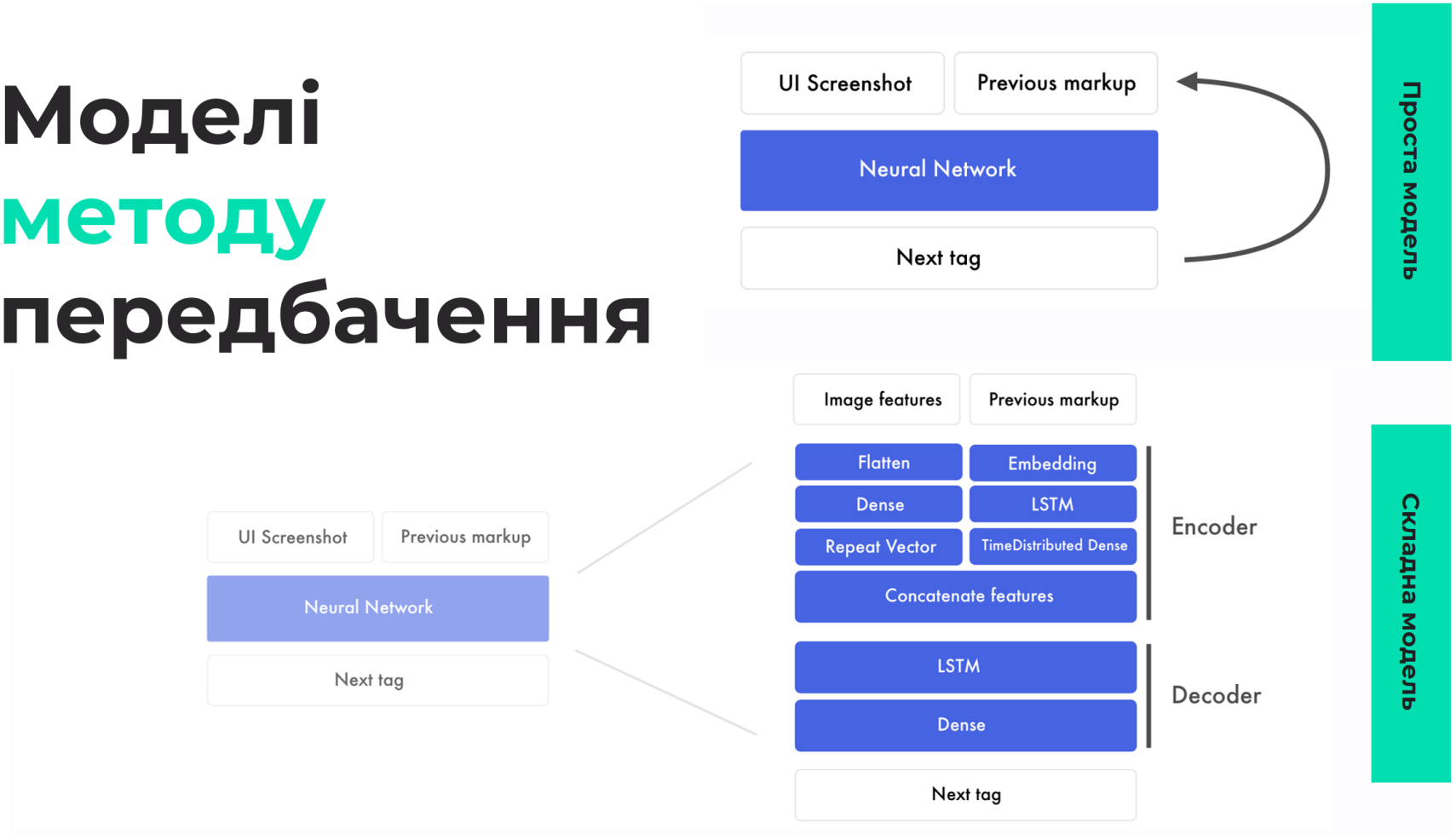
DSL

DeepCoder

- Своя DSL мова
- Високий вхідний бар'єр
- Обчислювальна потужність
- Відсутність користувацького інтерфейсу
- Погано розпізнає зображення
- Розпізнає до 5 рядків коду

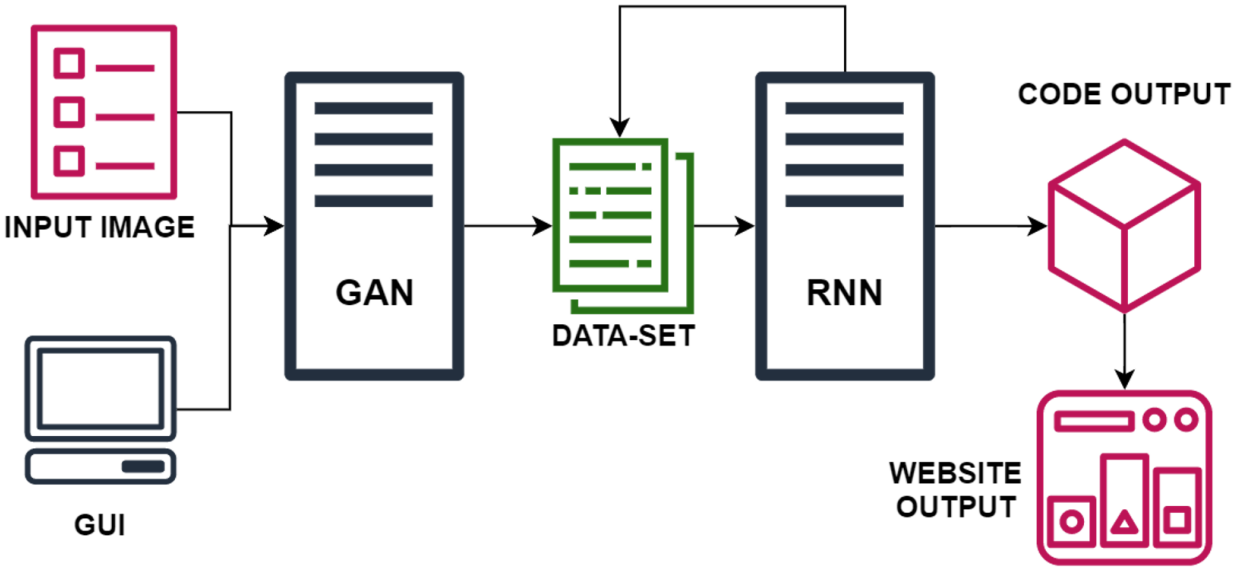
Недоліки

Моделі методу передбачення



12

Модель власного методу



13

Власний метод

Початок

.01

Створення вхідного зображення

Користувач створює макет сайту в редакторі

.02

Завантаження шаблону

Користувач завантажує макет до GUI

.03

Налаштування обробки

Користувач вносить свої налаштування

.04

Декодування вхідного зображення

GAN декодує вхідне зображення та створює дата-сет

.05

Аналіз дата-сету

RNN отримує дата-сет від GAN та починає генерацію коду

.06

Повторний аналіз

RNN перегенеровує код після ітераційного оновлення дата-сету

Основний етап

.07

Створення адаптивності

RNN перетворює згенерований код в адаптивний код

.08

Створення семантичності

RNN додає семантичну розмітку

.09

Створення структури проекту

Створюється структура проекту та прописуються залежності

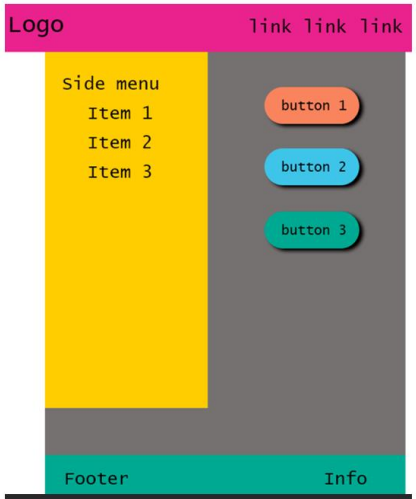
.10

Завантаження готового результату

Користувач може завантажити готовий результат або переглянути на локальному сервері

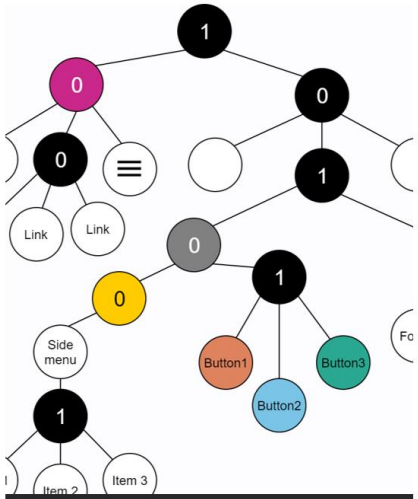
Фінальний етап

Процес роботи методу



Вхідне зображення

Користувач



Генерація дерева

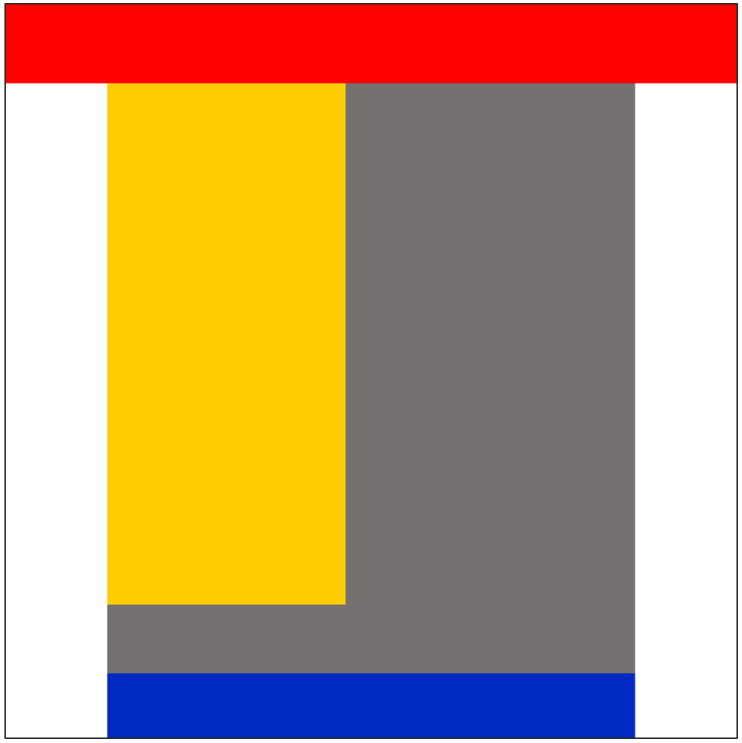
Метод

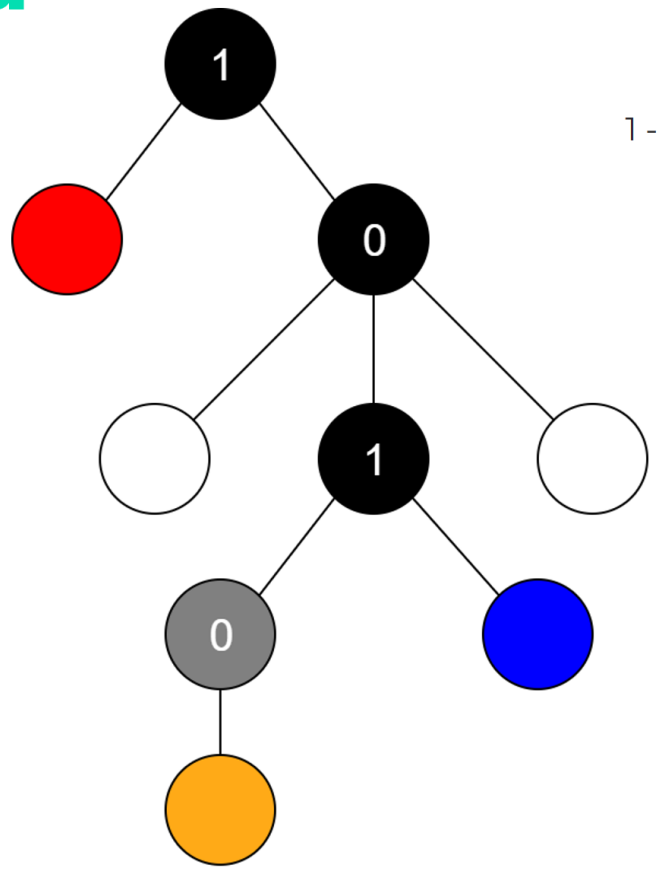
Генерація коду

Метод

15

Створення дерева

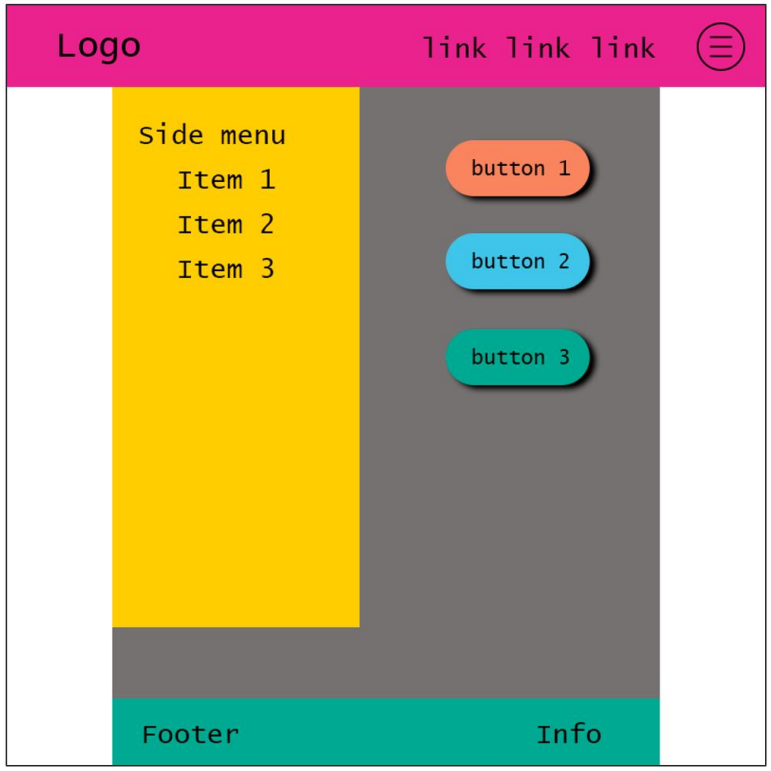


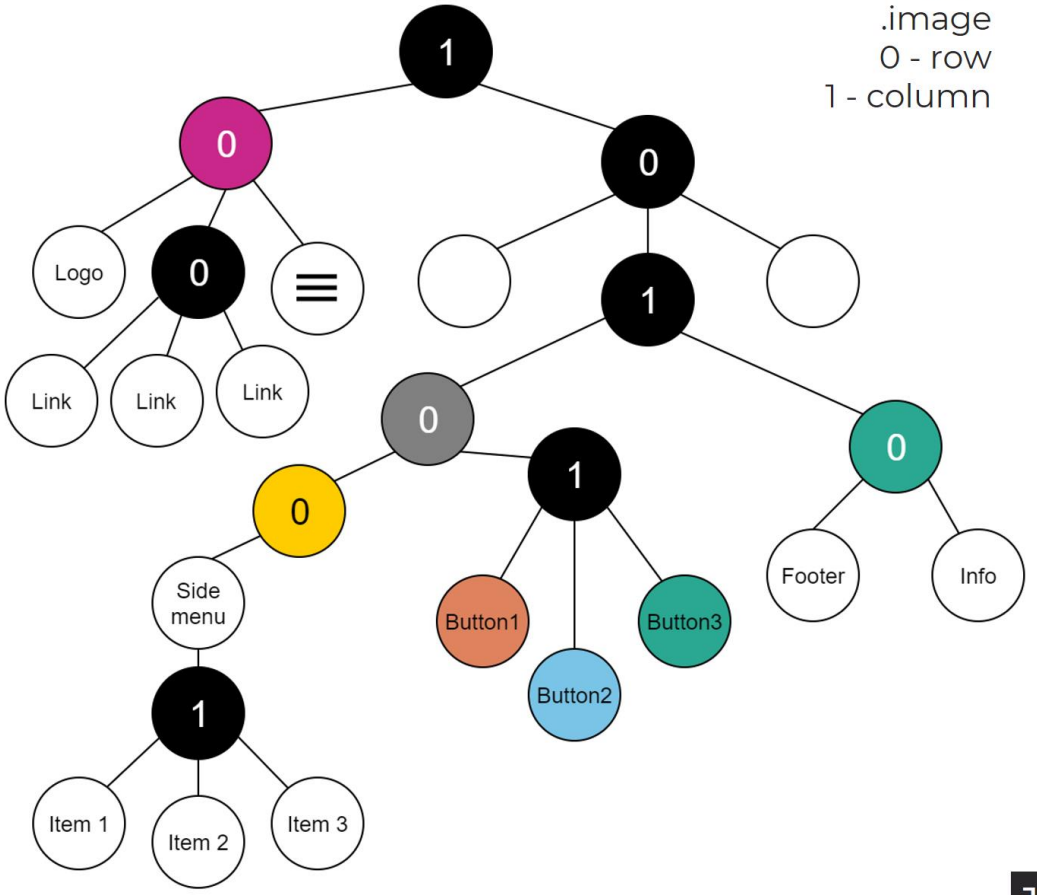


* .node
.text
.image
0 - row
1 - column

16

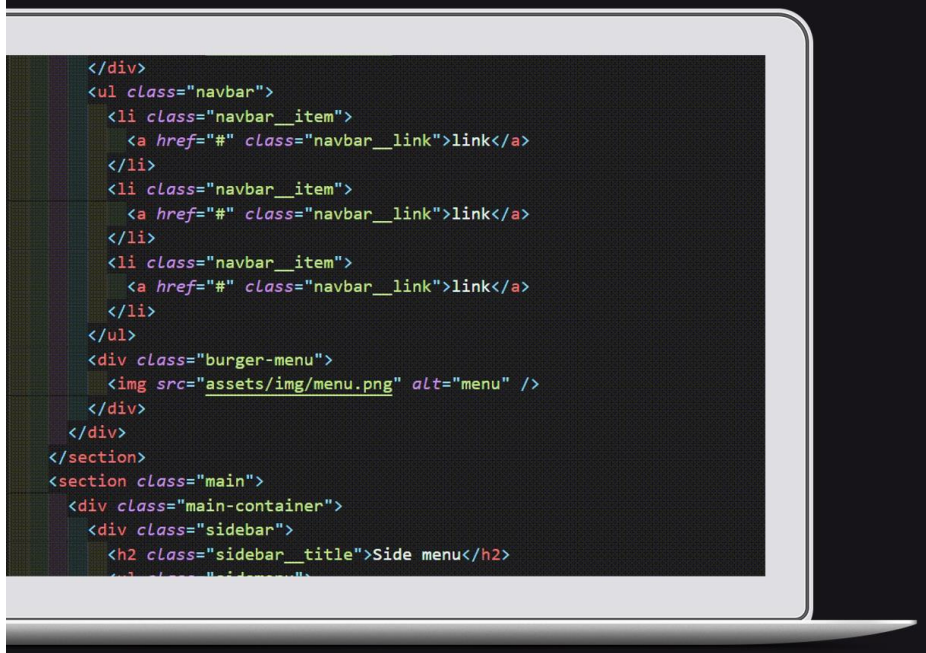
Створення дерева .02





* .node
.text
.image
0 - row
1 - column

17



Генерація коду

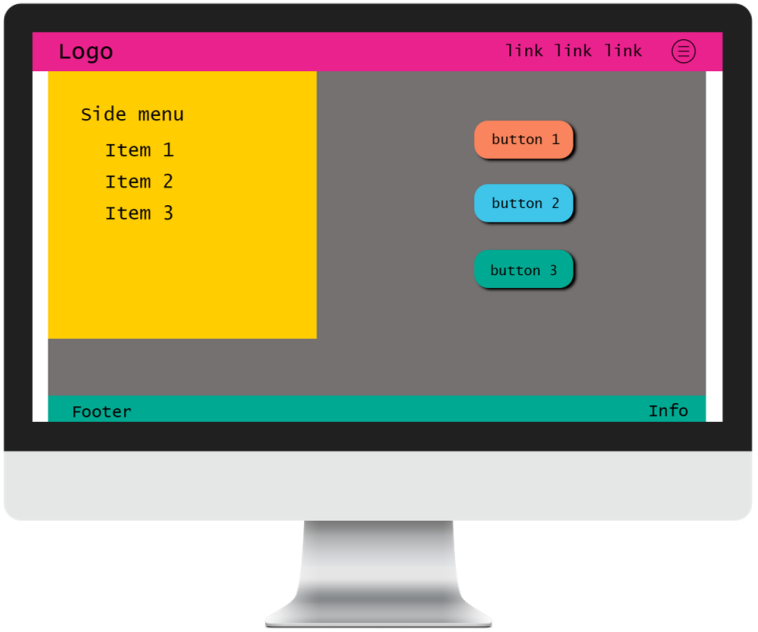
- ✓ RNN генерує HTML & CSS код на базі створеного GAN дерева.
- ✓ Одразу після цього створюється структура нового проекту та прописуються всі залежності в ***package.json***
- ✓ На цьому етапі можна запустити локальний сервер та переглянути результат роботи методу в різних браузерах та на різних пристроях.

18

Семантичність & Крос-браузерність

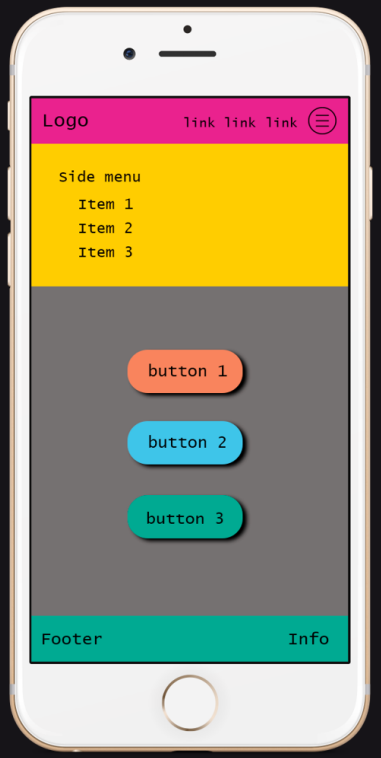
- Максимальна семантичність досягається використанням дата-сету утвореного на основі стандартів створення веб-сторінок **2019р**.

- Крос-браузерність досягається шляхом генерації властивостей з префіксами постачальника, які є запропонованими механізмами рендерингу сучасних браузерів.



Адаптивність & швидкість

- Адаптивність досягається шляхом використання власної сітки **GRID** та **FlexBox Layout**
- Згенерований програмний код успішно проходить тести **GooglePageSpeed**



Тестування методу

.01

Unit-тести

Запуск тестів відбувався на кожному етапі розробки

.02

Manual-тести

Ручне тестування було проведено на фінальному етапі

.03

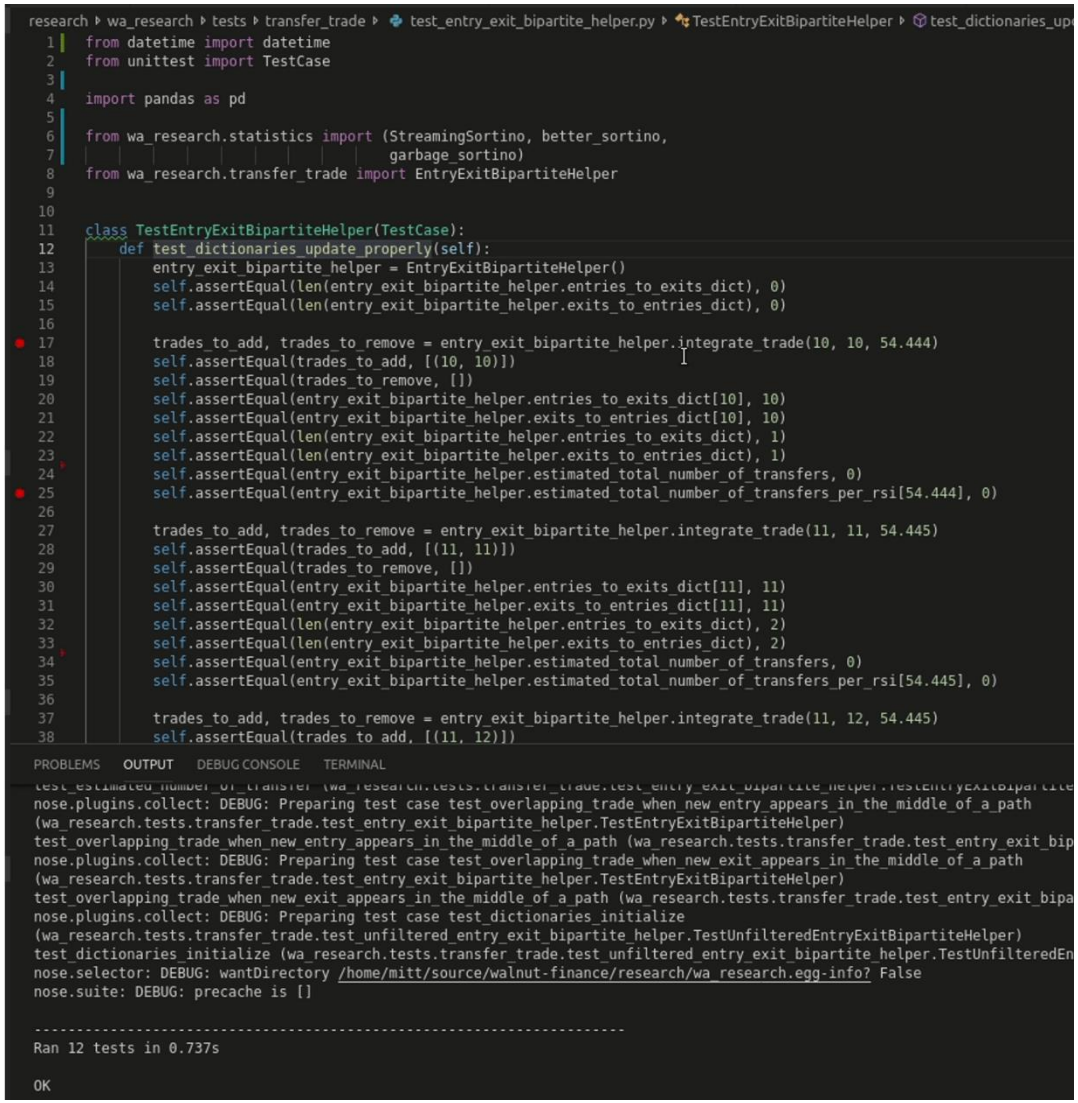
ESLint

ESLint було інтегровано на початку створення проекту

.04

Travis CI

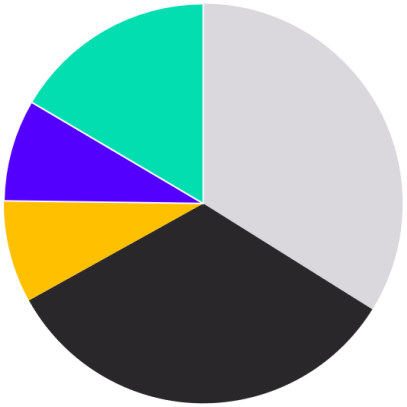
Використовувався разом із GitHub для безперервної інтеграції коду



Розподілення часу на виконання методу

Pix2Code

DeepCoder



Власний метод

- Аналіз зображення
- Генерація дерева
- Генерація коду
- Створення адаптивності
- Створення семантики

Порівняння результатів дослідження

Час роботи методу	Метод DEEPCODER	Метод PIX2CODE	Власний метод
Текст	2.980 s	2.454 s	2.208 s
Текст & зображення	6.142 s	5.754 s	5.178 s

Таблиця 1

≈ 10% ШВИДШЕ

Результати тестування методу

Завдяки:

- одночасній роботі GAN & RNN вдалося досягти прискорення загального часу роботи методу в середньому на **10 %**.
- використанню flex-box layout та відмові від Bootstrap, вдалося досягти адаптивності коду, чого немає в аналогів.
- власному дата-сету вдалося згенерувати максимально семантичний код, який успішно проходить тести **ESLint** та **GooglePageSpeed**.
- використанню **ReactJS** було створено зручний GUI, в якому можна налаштовувати роботу методу.

≈ 10% швидше	На Bootstrap	застаріла	відсутнє в аналогів
Загальний час роботи	**Адаптивність**	**Семантичність**	**Налаштування**

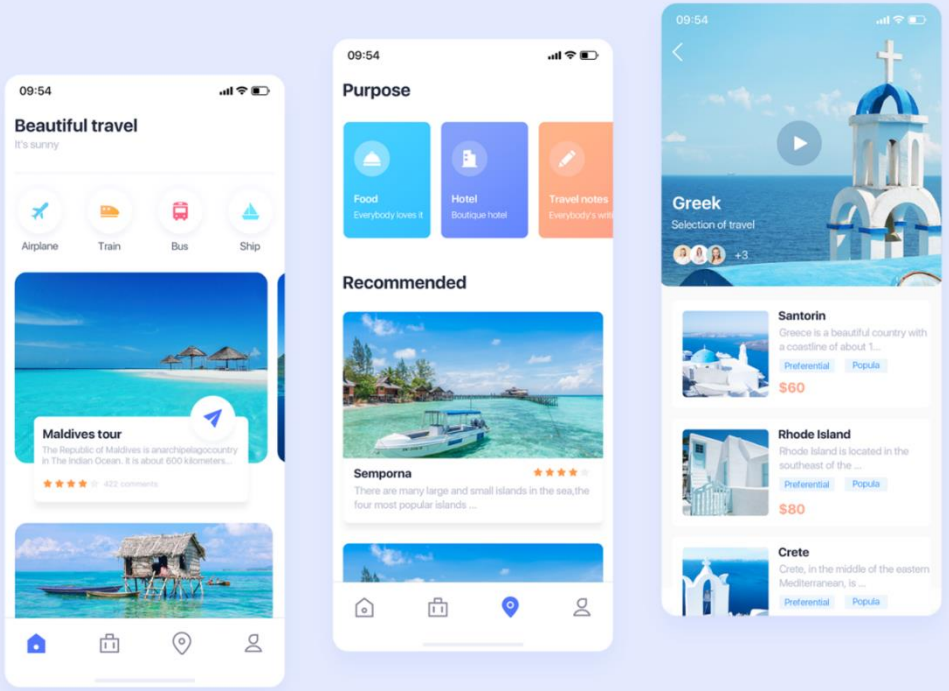
Використані технології

- Python
- HTML & CSS
- JavaScript
- ReactJS
- Redux
- NodeJS
- Git
- ExpressJS
- AWS
- Flask
- Mocha
- Travis CI
- Prettier + ESLint

25

Подальше використання

- Створення багато-сторінкових веб-сайтів
- Створення ReactJS застосунків
- Створення застосунків на платформах iOS та Android на мові Dart за допомоги фреймворка Flutter
- Загальна оптимізація
- Створення нових функцій



Наукова новизна

Запропонований власний програмний метод створення веб-сторінки за допомогою нейронної мережі на базі вхідного зображення відрізняється від існуючих методів способом створення унікального дата-сету, завдяки якому дозволяється одночасна робота GAN та RNN, що в свою чергу робить алгоритм швидшим від аналогів, та дозволяє генерувати адаптивний, семантичний та крос-браузерний програмний код веб-сторінки.

Бізнес-модель

Проблема	Рішення	Унікальна ціннісна пропозиція	Споживачі	Ключові метрики
• Немає простих, не ресурсоємких рішень; • Високий вхідний поріг знань для використання рішення; • Велика затратність; • Відсутність технічної документації; • Мала швидкодія.	Програмне забезпечення, що створює веб-сайт на базі вхідного шаблону-зображення за допомогою нейронної мережі та методу глибокого машинного навчання.	• Розроблений програмний продукт, що враховує всі потреби зацікавлених сторін. • Зменшення часу потрібного на рутинне створення шаблонних сайтів.	• Великі компанії • Розробники • Міні-сервіси • Звичайні користувачі.	• Кількість проданих ліцензій • Кількість зроблених мікротранзакцій.

Канали	Структури витрат	Потоки доходів	Ключові метрики
• Через відділи продажу та надання послуг.	• Початковий персонал для розробки програмного продукту, • Утримання персоналу для надання технічної підтримки • Заробітня платня та соціальні виплати • Оплата за оренду офісу • Оплата комунальних послуг.	• Доходи від продажу ліцензій • Доходи з мікротранзакцій.	• Кількість проданих ліцензій • Кількість зроблених мікротранзакцій.

Фінансовий план стартапу

Доходи(тис.$)	Продаж ліцензій	Зроблені мікро транзакції	Довготривалі підписки	Всього	
Сумарно за рік:	200	220	120	540	
Витрати(тис.$)	**Технічна підтримка**	**Оплата праці**	**Оренда та комунальні витрати**	**Реклама**	**Всього**
Сумарно за рік:	8	195	16	12	231

29

Висновки

- З’ясовано способи створення програмного коду на базі вхідного зображення за допомогою нейронної мережі, а саме RNN, GAN та NN
- Проведено аналіз варіантів покращення існуючих методів, на базі якого створено власний метод.
- Розроблено власний метод, загальна швидкодія якого покращена за рахунок одночасної роботи GAN та RNN.
- Створено зручний користувацький інтерфейс. Створено ряд налуштувань роботи методу, що вбудовані в GUI.
- Виконано тестування та порівняння результатів дослідження. Прискорення швидкодії загального часу роботи методу на 10%. Створенно спосіб генерації адаптивного, семантичного та крос-браузерного програмного коду.
- На основі результатів магістерської дисертації розроблено стартап проект.

Апробація

1. XI наукова конференція магістрантів та аспірантів «Прикладна математика та комп’ютинг» (ПМК-2019-2).

Дякую за увагу !